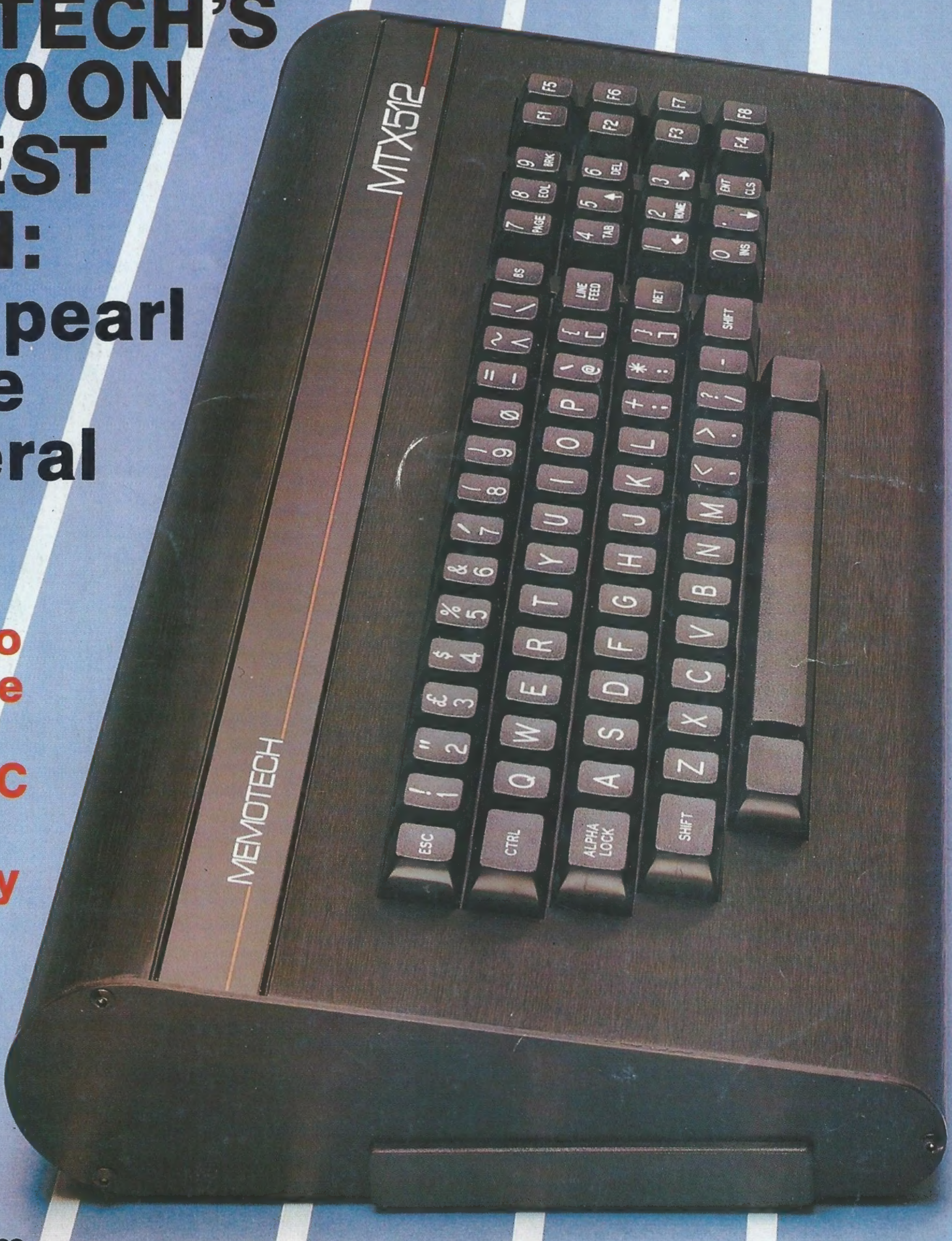
An Argus Specialist Publication

# Computing today

APRIL 1984
85p

## MEMOTECH'S MTX500 ON THE TEST BENCH:

A black pearl from the peripheral people




Learn how to pull machine code apart with our BBC OS1.2 ROM disassembly feature

Have a Brain Storming session on your CP/M system

More of the same – Recursion on four Apple languages


# CONTENTS

## VOL 6 NO 2 APRIL 1984


**EDITORIAL & ADVERTISEMENT OFFICE**
**No. 1, Golden Square, London WC1 3AB.**
**Telephone 01-437 0626. Telex 8811896.**

**Editor:**
Peter Green
**Editorial Assistant:**
Jamie Clary
**Senior Advertisement Manager:**
Malcolm Wynd
**Advertisement Copy Control:**
Sue Couchman,
Ann McDermott
**Managing Editor:**
Ron Harris BSc
**Chief Executive:**
T.J. Connell

**Origination by Design International.**

**ABC Member of the Audit Bureau of Circulation**

ISSN 0142-7210

Computing Today is normally published on the second Friday in the month preceding cover date. Distributed by: Argus Press Sales & Distribution Ltd, 12-18 Paul Street, London EC2A 4JS. 01-247 8233. Printed by: Alabaster Passmore & Sons Ltd, Maidstone, Kent.





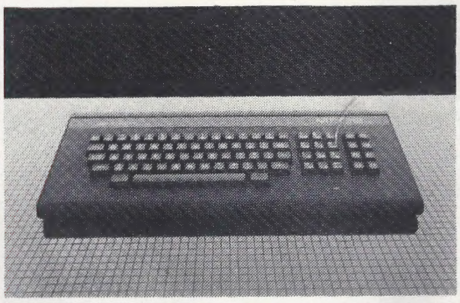
Subscription rates: UK £13.30 including postage. Airmail and other rates upon application to Computing Today Subscriptions Department, PO Box 35, Wolsey House, Wolsey Road, Hemel Hempstead, Herts HP2 4SS.


**Computing Today is constantly on the look-out for well written articles and programs. If you think that your efforts meet our standards, please feel free to submit your work to us for consideration.**

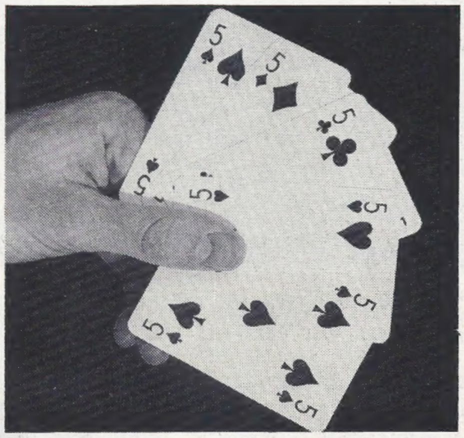
**All material should be typed. Any programs submitted must be listed (cassette tapes and discs will not be accepted) and should be accompanied by sufficient documentation to enable their implementation. Please enclose an SAE if you want your manuscript returned, all submissions will be acknowledged. Any published work will be paid for.**

**All work for consideration should be sent to the Editor at our Golden Square address.**









# NEWS

## ON COURSE DOWN SOUTH

A new, private computer college threw open its doors to the press this month, to provide an insight into its new series of courses, which, by the time you read this, will already be in full swing. The Southern Computer College, based in Surrey, offers a range of standard courses designed to teach the rudiments of programming, and to allow potential users 'hands-on' experience in a typical office environment. The college also offers facilities for developing specialised courses in response to specific requests from interested companies.

The SCC's principal, Mr. S. J. Solomon, has considerable teaching experience, having spent many years in a variety of colleges, and apart from his work with the SCC, enjoys considerable involvement with the Open University's computing faculty. The teaching facilities inside the college are very satisfactory, and has amongst other things a 'teaching laboratory' — a lavishly carpeted classroom housing several Olivetti L1's and a Sirius, plus the usual array of peripherals.

The machines are capable of running a fair range of software, each package selected to represent the kinds of applications that the students are likely to have contact with at work. Facilities for specialised teaching based around tailored packages can also be catered for. Interested companies are already being invited to 'open days', allowing company reps a chance to observe the surroundings, and to see for themselves exactly what the college has to offer.

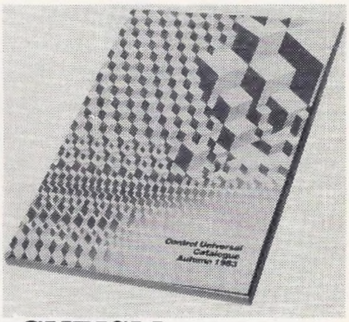
Mr Solomon has gone to some length in attempting to illustrate the purpose of the college — "I have tried to continuously develop our aims in response to the suggestions put forward by those who have visited us" he reported. Interest is expected from many businesses in and around the Surrey area, although the 'Southern Computer Colleges Ltd.' name does suggest that if the project is a success, then expansion can be expected at some later date.

The general course comprising eight weekly sessions, each of three hours duration, costs £168 + VAT, and as mentioned, the college can offer more specialised courses in response to demand. For further information contact Southern Computer Colleges Ltd, Capitol House, 662, London Road, North Cheam, Surrey.

## SPINE OF THE TIMES

Hot on the heels of the much-vaunted mouse for cursor control and menu selection on microcomputers comes yet another technical innovation designed to make computer use effortless. **Computing Today** can exclusively reveal the development of a new input device called the Hedgehog by its designer, British engineer Douglas Dinsdale.

The new device consists of a small hemisphere covered by many small movement sensors. Each sensor projects a fine 'bristle' outwards, giving the Hedgehog its unique appearance and making its name inevitable. The Hedgehog's principle of operation is based on that of the human skin, with



## CUBISM

Control Universal of Cambridge have just released the much awaited CUBE catalogue — one hundred and fifty pages of photographs, specifications and technical information which describe the extensive CUBE range of Eurocards, rack-mounted computers and peripherals — which now forms the most comprehensive selection of modular hardware offered by any British manufacturer.

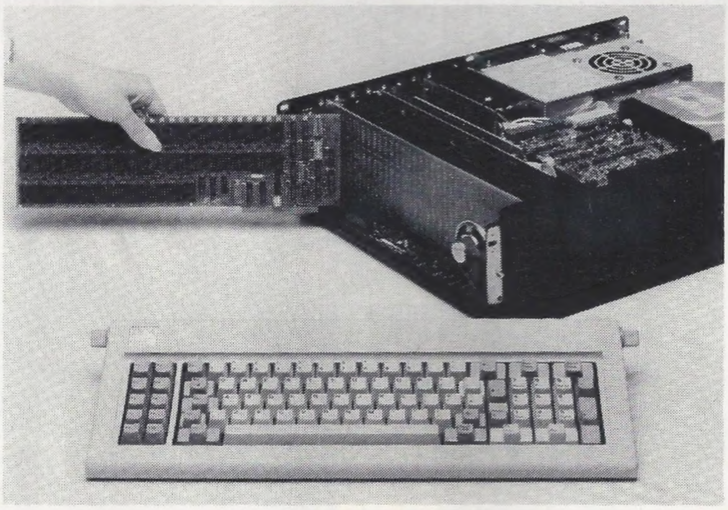
The catalogue's increased contents reflect the phenomenal growth at Control Universal over the past 12 months. After developing a prominent position in the field of industrial control, the company has now grown into a full-scale manufacturer and distributor.

Development systems based on Eurocards have always been a speciality of the company. They produce direct replacements (which in most cases are enhanced versions) of the currently unavailable Acorn range of Eurocards. Since the last catalogue, Control Universal's research and development department have produced a large number of

## RAMMING THE IBM

Advanced Peripheral Products Ltd, the Feltham based distributor of microcomputer upgrades, has announced the introduction of the JRAM memory card. This memory card increases the internal memory capacity of the IBM-PC beyond the 640 Kbytes barrier.

Under an agreement with Tall Tree Systems of California, Advanced Peripheral Products offer memory expansion cards in 512 Kbyte increments for the IBM-PC, XT and compatible machines, that will allow the user to add up to four JRAM cards into the system. Together with the comptuer's existing 512 Kbytes, this gives a powerful 2.5 Mbytes of random access memory.



The JRAM card has a maximum access time of 275 nanoseconds (Ns) and uses a unique hardware pager in the form of a small high-speed memory chip controlled by software. This allows each 64 K band of memory to be referenced through any 64Kbyte address boundary accessed by the CPU.

Shipped unchanged since March 1982 within the USA, the JRAM card enters the European market with a proven track record and is now available from Advanced Peripheral Products Ltd. For further information contact Advanced Peripheral Products Ltd, Enterprise House, Central Way, North Feltham Trading Estate, Feltham, Middx TW14 0RX (telephone 01-844 1200.

its many touch receptors buried beneath the epidermis and receiving their input via the hairs which sprout from them. In the same way that the long moment arm of the hairs increases sensitivity of the skin receptors to small environmental changes, the Hedgehog's bristles give it an unsurpassed ability to sense vibrations. Dinsdale claims that it is possible to move a cursor freely over the VDU screen and toggle menus merely by lightly brushing your hands across the Hedgehog.

Of course it isn't quite as simple as this: a great deal of decoding is necessary to determine exactly what action is required from the many input signals being generated at any given moment, and a small single-chip computer based on the ubiquitous ULA lies at the heart of the Hedgehog. At present this means tht the Hedgehog is not competitive with the various Mice available, but Dinsdale hopes that mass-production will eventually bring the price down and win industry acceptance.

Frankly, we're somewhat sceptical of the whole idea here. If the Hedgehog is so sensitive, surely you'll get erroneous inputs from draughts and doors banging? It would be pretty distressing to have the file you've just spent three hours working on wiped from disc because someone opened the office window. Ingenuity is all very well, but we feel that Douglas and his research company, S. Norman Associates are barking up an evolutionary dead-end.

## SECONDS OUT FOR BBC

Torch Computers have announced the release of their 8-bit second processor (the ZEP100) for the BBC Model B Microcomputer. Previously the Z80 processor was only available as part of the Torch Z80 Disc Pack. Now the ZEP100 can be used to upgrade a Model B BBC, with disc interface and any compatible 400K disc drives, to a complete business microcomputer.

Running the well-established CPN operating system, the ZEP100 opens the door to the vast library of CP/M applications programs and languages for use on the BBC. The user can switch back to running standard BBC programs.

A complete business software package, including word processing, spelling checker, spreadsheet and database filing systems are supplied free of charge with each ZEP100. Also supplied on disc are several utility programs, a Music Compiler, and a character design program.

The full TORCHNET operating system is supplied in EPROM so that a Model B BBC with an Econet interface can be used as a powerful workstation within a Torch Network. Several BBC comptuers could be upgraded in this way and networked into one Torch Hard Disc machine to provide an inexpensive and highly effective local area network.

The cost of the ZEP100 is £375. For further information, please contact Torch Computers Ltd, Century House, 19 High Street, Marlow, Bucks.

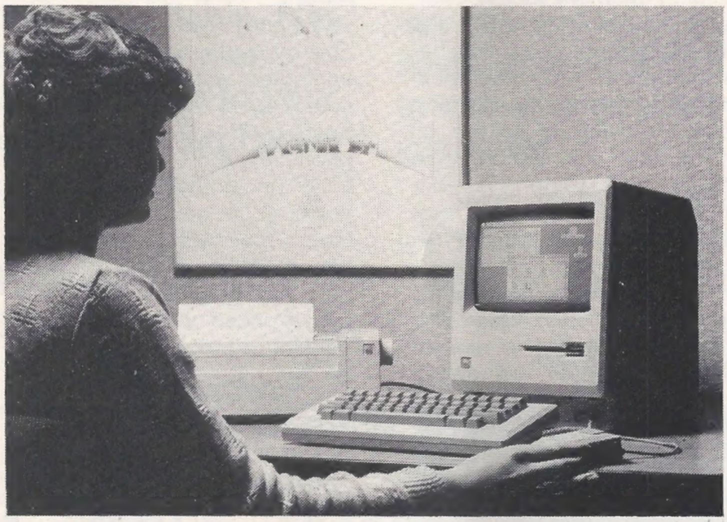
innovative products to suit the cost-conscious and developing needs of the industrial marketplace.

Everything from "EuroCUBE" single board controllers to complete "CUBE" systems are available through the catalogue. "Delegate/UniCUBE" industrial and laboratory control modules (part of the new range) are described in depth, as are the multiplying range of add-ons, software, and extensions for the BBC micro. Of particular interest are "BEEBEX", which extends the BBC to enable it to run all the CUBE modules, "CU-GRAPH", a high performance VDU interface, and "EuroBEEB", a CPU card.

Design and development aids are given greater prominence in this catalogue. The FLEX Operating System, for instance, is an easy-to-use software tool for the 6809 processor which allows not only high level languages such as Pascal, BCPL, FORTRAN and FORTH to be run on a low-cost CUBE system, but a range of efficient cross-compilers too.

Also included are BBC micros and accessories, printers, VDUs, discs, ICs, software and stationery. A reply card enclosed with the catalogue invites users to send for microprocessor instruction cards and a BBC BASIC reference card free of charge.

The new CUBE catalogue is available free from: Control Universal Ltd, Unit 2, Anderson's Court, Newnham Road, Cambridge CB3 9EZ (telephone 0223 358757).



## MICROSOFT AND MACINTOSH

Microsoft Corp announces that it is releasing four applications packages for Apple Corp's newly-announced Macintosh microcomputer. The applications include enhanced versions of Multiplan and Microsoft Word, and two new applications — Microsoft File and Microsoft Chart. Microsoft will also be making its BASIC available for the Macintosh.

Microsoft has chosen Apple's launch of its new computer as the time to release two new applications programs onto the market. These are a database management program, Microsoft File, and a graphics program, Microsoft Chart. All Microsoft's applications offered for the Macintosh make full use of the hardware and software concepts employed by Macintosh — a mouse and high-resolution graphics screen, 'pop-up' menus and icons.

**Microsoft Multiplan** on the Macintosh provides all the features of other versions of Multiplan with additional enhancements. An 'UNDO' command allows reversal of the last change to the spreadsheet. Recalculation of data is now much faster, and printing options have been extended to include headers and footers and automatic page numbering. Multiplan also employs Macintosh's 'cut and paste' concept, allowing data to be transferred between two different spreadsheets. Files produced with Multiplan on other microcomputers are compatible with Macintosh Multiplan.

**Microsoft Word**, Microsoft's word processor, uses the full graphics capabilities of the Macintosh screen to allow full visual representation of text and graphics on the screen. This includes proportional spacing and support for Macintosh's character fonts. Moving and copying portions of text, including moving between documents, is accomplished with the standard Macintosh edit functions of 'cut', 'copy' and 'paste'. Microsoft Word also includes an extensive facility for merging information from other documents — or Microsoft File — into form letters.

**Microsoft Chart** is a business graphics program that can display information in graphical form. Chart can format graphs in any of a number of different formats — pie charts, line graphs, bar graphs, scatter diagrams and histograms. Any chart can be quickly reformatted into a different format. Data can be drawn from an existing Multiplan file, or may be entered, edited and formatted directly in windows on the screen. With Chart, the user has full control over the graphical display. Fonts, type size, position and other features can be easily changed using the mouse.

**Microsoft File** is a database management program using forms to enter and view data. Microsoft File can then be used to supply data to any application that can read and write text files, or to Macintosh's clipboard.

**Microsoft BASIC** takes full advantage of the addressing capability of the 6800 microprocessor employed by Macintosh, and includes a decimal maths package with 14-digit precision, and string variables

and string expressions of up to 32,767 characters each. Microsoft BASIC fully incorporates the Macintosh interface, and presents the user with up to three kinds of windows — one for command entry or editing a listing, one for viewing the program listing and one for the output of a running program. BASIC also provides many of the extended graphics capabilities of Microsoft's GW-BASIC and has access to the Quickdraw routines supplied with the Macintosh. The BASIC also supports Macintosh's font manager.

The BASIC is source code compatible with all standard versions of Microsoft BASIC, allowing for easy migration of programs written in Microsoft BASIC to the Macintosh.

## SPECTRUM SOUND BOOSTER

Zeal Marketing have designed and are now producing a sound booster for the Sinclair Spectrum computer. The unit, which is attractively packaged, comes complete with leads and a load-save facility which obviates the need for constant plugging and unplugging. A hefty 3" loudspeaker is incorporated and a volume control for the more than adequate sound input. The device requires no batteries and all connections to the computer are made externally, thus ensuring no invalidation of the guarantee.

At £14.99 the unit is competitively priced and provides a major enhancement to Sinclair Spectrum users: it is available direct from the company. Full details are available to readers on request: for further information contact Zeal Marketing Limited, Vanguard Trading Estate, Storforth Lane, Chesterfield S40 2TZ (telephone: 0246 208555).

## SHIPMAN-SHAPE SOFTWARE

Tom Shipman (Supplies) Limited has recently released four new educational programs for use with the BBC computer. One offers a valuable aid to those who need help in preparing and revising for biology examinations, while the other three combine the interest of an arcade game with the instructional value of a test to assist

children in learning about words, numbers and geography.

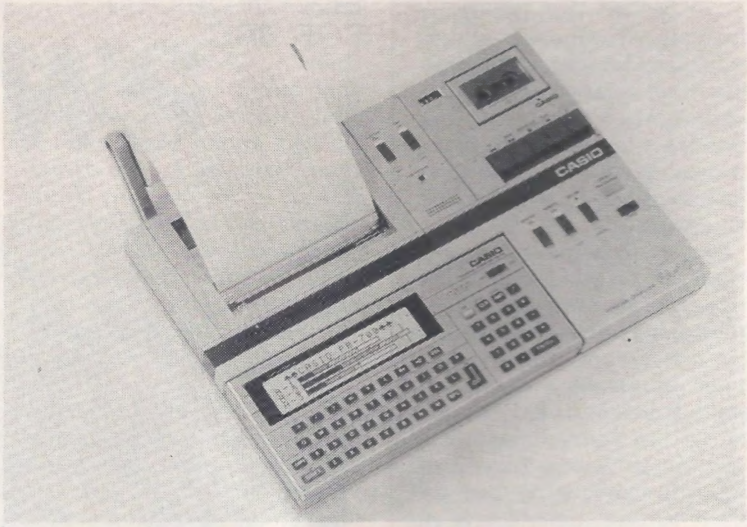
All of these programs have been designed with the close involvement of young people who have recently finished their studies. By employing the generation which has grown up using computers, and who understand the problems and demands faced by their contemporaries, the company feels these programs will attract and maintain the interest of children in a way which many software manufacturers have failed to do.

Europe Rally (for children from 9 - 14) is an interactive motor racing game and geography test which gives children the thrill of driving a racing car and teaches them about the various European countries at the same time. They travel along roads inter-connecting European countries, brought to life by full colour graphics. Before entering a new country, they are forced to stop at a barrier and answer several questions about it before proceeding. Each correct answer is rewarded with a tune.

Magic numbers (7 - 11 years) is a comprehensive program in several sections, which uses sound and colour graphics to help a child learn basic addition, subtraction, multiplication and division. The "counting" section requires the child to count the number of objects on the screen. There are a multitude of different shapes, in varying colours, to hold a child's interest.

"Count sums" uses the objects in "counting" to progress to simple addition, subtraction, multiplication and division. Each correct answer receives a tuneful reward. In the final section, "Magic Numbers", an arithmetical problem is displayed with one of the numbers missing. To complete the sum the child plays a game in which they steer a grabber to select the correct number from a batch moving along the screen.

Each section features full colour graphics and sound effects. As a bonus, there is



## WIPING YOUR DATA

Warnes Data Wipers are a simple solution to the programmers retitling problem facing computer buffs. Instead of the illegible mess you get with the pencil/eraser combination (which may harm the disc surface) or trying to scrape off redundant information, the Warnes Data Wipers kit enables you to mark each program simply and clearly. When you want to make an alteration you wipe the label clean with a slightly dampened tissue and start again. The kits are reasonably priced and include 45 specially designed labels and a special Wipers pen. Computer labels are available in four sizes — cassette, floppy disc 3¼"/5¼" and 8", and platter.

Presently, Data Wipers are only available to the personal computer user by mail order from Warnes Wipers, 23 Werter Road, London SW15 2LL (telephone: 01-788 1782) priced at £3.49 (including postage and packing). However, from early March they will be available from branches of W.H. Smiths (that sell computers) in smaller packs, priced under £2.00 for a pack of 12 labels.

## COMPUTER IN YOUR POCKET

The start of a progressive push into the personal computer market by the world's biggest pocket calculator manufacturer is marked by Casio's launch of model PB700.

Using four penlight torch batteries for up to 100 hours of continuous use, PB700 measures 7⅞ x 3½ x ⅞ inches and weighs only 11.1 ounces, yet it has 4K RAM, expandable to 16K RAM in 4K steps. A Qwerty layout keyboard and a 4 line by 20 character LCD display are included in this small package. The display can also show 160 by 32 dot graphics.

Using an extended BASIC the keyboard can be used to enter upper and lower case characters, BASIC keywords, symbols, and numbers on the separate numeric keypad. The PB700 is supplied with a comprehensive instruction book, quick reference guide and carrycase.

To enhance its performance Casio are also introducing the FA10 cradle. This is A4 size and has the following facilities: a four colour plotter/printer using 4½-inch plain paper, an external cassette port with remote on/off and a section to take the optional CM1 micro cassette deck.

FA10 is supplied with a small carrycase, two rolls of paper, four colour pens, external cassette leads and an instruction book. CM1 is supplied with a demonstration tape and instruction book. The complete package weights under 5 lb (2½ kg).

Recommended retail prices are: PB700 (4K) £139.00; FA10 printer/plotter £189.00; OR4 (4K RAM) £29.95; CM1 micro cassette £65.00. All prices include VAT. Sales enquiries to Casio Electronics Co Ltd, Unit Six, 1000 North Circular Road, London NW2 7JD (telephone 01-450 9131).

# OUT NOW! 811 IDEAS FOR YOUR COMPUTER

NEW SAME-DAY DELIVERY IDEAS BOOK

In the Inmac catalogue of ideas for your computer check to see what is in the latest issue for you:

**Cables, cables and more cables!** How to order custom-built cables. When to use screened cables. How to choose the right **EIA RS-232** extension cables for your VDU's and how to connect your **Apple, Commodore,** or other micros to **Qume, Diablo, NEC** printers. Compatible cables for connecting new peripherals to your **DEC, Data General, Hewlett Packard, IBM, Osborne** and **Tandy computers.**

**Lifetime Floppies** Read about Inmac Plus — the floppy with a lifetime guarantee! Choose from our range of **100% error-free** disk cartridges, **flag-free** disk packs, each with a 3-year replacement guarantee.

**New Ideas!** Dozens of exciting new products — many never seen in Europe before now — and lots of helpful hints on getting more from your computer department.

**Service and Quality — a promise you can rely on!** We promise a **next day** delivery service of our products, a **30-day risk-FREE trial**, and a **12-month** trouble-free quality replacement guarantee.

Send today for your **FREE** catalogue with no obligation to purchase — <u>but be warned:</u> you will want to buy once you have read the Inmac catalogue ideas book!


**FREE! IF YOU USE A COMPUTER ☎ 09285-67551/01 740 9450**

Send this coupon to
Dept. 661, Inmac (UK) Ltd., Davy Road, Astmoor, Runcorn, Cheshire WA7 1QF.

Name________________

Company________________

Title________________

Address________________

________________

Postcode________ Tel.________

Computer System________________

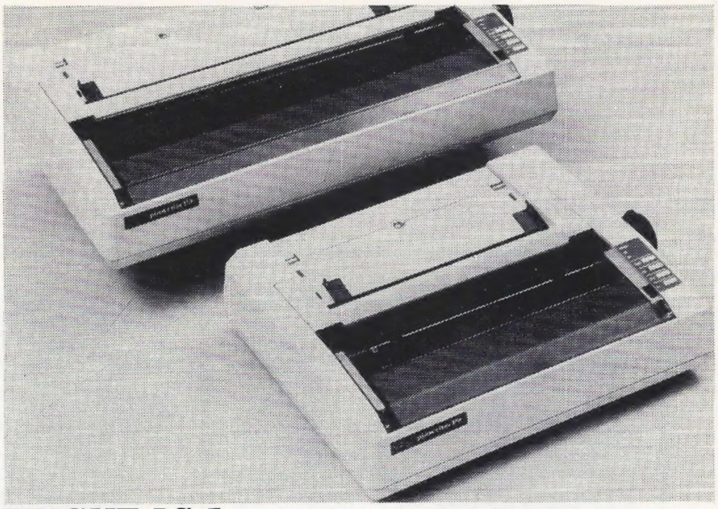
24 hour tel. no.
☎ 09285-67551

rating on the performance of the child at the end of each section, which allows parents or teachers to see how they are progressing.

Word Chaser (7 - 14 years) is a colourful, graphic, educational program, in which the child is presented with a sentence with one of the words missing. They can guide a figure through an obstacle course of moving gates, and, having accomplished that, choose a word (out of three possible choices) that completes the sentence correctly. If the word is correct, there is a tuneful reward. If the word is wrong, the child will be shown the word that fits, then the complete sentence. The sentences are grouped in files, separate from the main program. Three files are supplied in the package, with words ranging from easy to difficult. The files are quick and easy to load, employing clear step-by-step instructions.

Biology (14 and above) is an interesting and informative program which will satisfy both 'O' Level students and those with an interest in the structure of the human body. There are two sections, organs and bones, both with colour high-resolution graphics, which describe and locate each organ and bone in turn on a detailed drawing of the human body. Included in the program is a test on both sections, involving one word and multiple choice questions, which employ the detailed body map as an illustration.

All of these programs are available from Tom Shipman (Supplies) Limited at Heron Trading Estate, Bruce Grove, Wickford, Essex SS11 8BP, at a cost of £7.95 including VAT.



## BRIGHT AS A NEW PIN

NEC are launching a new three speed matrix printer — the Pinwriter. Designed to complement the highly successful Spinwriter range, the Pinwriter offers the user the option of three print speeds — near letter quality (30 cps), high density (90 cps) and draft quality (180 cps). Two models are available, the P2 (80 columns) and the P3 (136 columns). Prices will start from under £700 for the P2 and £800 for the P3.

Commenting on the new product, Colin Zardin, NEC's European Sales Manager for printers and peripherals said, "Whilst there are other multimode matrix printers on the market they tend to be rather expensive, generally costing over £1500. The Pinwriter is approximately half that price. With one cost-effective printer NEC now covers all the user's printing requirements for word processing, graphics and high speed output. We expect it to

## ANAGRAM'S LEDGER-DEMAIN

As flexible as the machine it is designed for, Anagram Systems' new Purchase Ledger for the Commodore 64 is in a class of its own when compared to similar business programs available for this truly hybrid business/home computer. Most other packages are programs written for for the VIC 20 micro and merely upgraded to run on the 64. But because Anagram Systems are specialists in producing high quality software for the strictly business-orientated Commodore 8000 series machines, their Purchase Ledger '64 package benefits from the sophisticated features of a program which has been adapted downwards rather than upgraded.

With the non-technical user in mind, the Purchase Ledger for the 64 operates in plain English, and is designed to cover a wide range of business requirements. Using suppliers' names, the ledger enables a comprehsive file of supplier accounts to be maintained. Invoice payments and debit notes can be posted to each one, and remittance advice printed, together with a usefully wide variety of internal reports and summaries, such as invoice, payment, discount and debit note lists, outstanding balances, invoices, statements, and customer names and addresses.

Analysis of purchases is possible under different nominal headings and open item accounts mean that each individual invoice stays on file until it is paid, with cleared invoices removed on a monthly cycle. Depending on how much information is stored for each supplier, a typical user will find that he can have between 150-175 supplier accounts with 50 nominal accounts having between 4-10 invoices outstanding per supplier.

For the small business which already has a Commodore 64, compatible disk drive and printer, the Purchase Ledger is available for only £75, yet performs the same range of functions as software costing 3-4 times as much for larger machines. What is more, it dovetails in with Anagram's Sales Ledger package and the Commodore Stock Control System, written by Anagram, to provide a total small business package.

Sales Ledger is also a version of a program written by Anagram for the serious business user of larger Commodore machines. Specially tailor made to run on the Commodore 64, it is now available to businesses which don't want the expense of a big computer package, yet which could profit by streamlining their sales recording system.

Customer account files, using their names and not numbers are maintained and operated on a monthly cycle, with paid invoices being deleted monthly, and outstanding accounts remaining on file until cleared, in this open item accounting based package. Invoices, payments and credit notes can be posted to each account, and Sales Ledger prints invoices, credit notes and statements. A handy 'suspense' account is also provided where unrecognised cheques or deposits can be held before being sent to the right accounts.

The flexibility of Sales Ledger allows for settlement and trade discounts, line by line, or as a percentage to be applied to goods totals. Building up a comprehensive picture of your sales profile at particular times is made easy with Sales Ledger's printed report functions, which cover a variety of areas such as current customer balances, payments and discounts taken, aged debtors, sales lists, suspense account listing, and customer names and addresses.

For just £75 Sales Ledger will transform a Commodore 64 computer into a sophisticated store of sales files. All that is needed is a compatible disc drive and printer. Sales Ledger and Purchase Ledger are available from your local Commodore dealer or direct from Anagram Systems, 60a Queen Street, Horsham, West Sussex RH13 5AD (telephone 0403 59551).

appeal to the mass market of business micro users as well as to DP management".

The Pinwriter has an 18-pin matrix head and each pin can be addressed individually, giving the Pinwriter excellent graphics capabilities. The Pinwriter's horizontal character spacing is software selectable and offers a choice of 10, 12, and 17 cpi (characters per inch) and proportional spacing. Eight international character sets are provided together with a down-line loan capability.

The new printers come with a choice of three plug-in interface modules — Centronics parallel, IBM PC parallel and RS232C serial. Each moduel has a 3.5K buffer. Forms handling options include tractor feeds, single cut sheet guides and automatic cut sheet feeders.

The Pinwriter P2 and P3 will be available in the UK and Europe from early March onwards. For further information please contact Norman Fox, NEC Business Systems (Europe) Ltd, NEC House, 164/166 Drummond Street, London NW1 3HP (telephone 01-388 6100).

## A MONITOR FROM ZENITH

Zenith Data Systems has launched a new colour monitor, the ZVM-133, which has the ability to display an infinite array of colours and intensities with sharp graphics. The screen has an extremely black background from which the colours stand out sharp and clear.

A high resolution colour graphics monitor, the ZVM-133 is designed for use with Zenith's own range of desktop computers, the Z100 series. In addition, it is compatible with most business microcomputers having RGB direct drive outputs, such as the IBM Personal Computer and Apple III.

The ZVM-133 has a 25 line by 80 characters display and pixel resolution of 640 dots by 480 lines (interlaced), which creates impressive graphics. It has a wide bandwidth of 20 MHz with the rise time of 70 nanoseconds to generate crisp lines, pure colours and intensely clear copy.

Easy to reach front-access user controls enable convenient operator adjustments. An LED indicator notifies the user when the monitor is operable. A "Green Screen Only" feature incorporated in this colour monitor, eliminates all other colours. This enables monochromatic material to be displayed on the black face of the screen.

DC-coupling permits the video display to retain its colour balance from a single dot to full screen data. The colour monitor has contemporary styling and a strong but lightweight modular chassis for increased portability.

The new ZVM-133 monitor is priced at £395.00 plus VAT, and attractive quantity discount packages are available. Zenith Data Systems Limited is at Bristol Road, Gloucester, GL2 6EE (telephone: 0452 29451).

## LOW-COST TERMINAL FROM COMART

Designed to provide a smart yet functional terminal in any office or factory environment, the comart WY50 is now available with any of the Communicator range of 20 different modular microcomputers. It is a low-cost terminal that provides a full 14-inch green screen with easily readable 80 or 132-column displays, but has a tiny 'footprint' of less than one square foot.

Costing only £595, the VDU sets new high standards in design, ergonomics and functionaility. The low-profile detachable keyboard has a 101-key typewriter layout with 16 programmable function keys, cursor control and numeric pad sections. Soft set-up of all parameters is entered from the keyboard while parameters are displayed at the bottom of the screen. The Comart WY50 also gives users the ability to emulate ADM-31, Hazeltine, ADDS, and Televideo terminals.

The Comart WY50 is available for all Comart Communicator systems, from the 8-bit single user system up to the largest multi-user, multi-tasking systems and local area networks. Comart are at Little End Road, Eaton Socon, St. Neots, Huntingdon, Cambridgeshire PE19 3JG (telephone 0480 215005: telex 32514).

## CRA HITS BACK AT BEEB

The following statement has been issued by the Computer Retailers Association:

In cooperation with the British Computer Society Copyright Committee, the Computer Retailers Association is taking various steps to publicise the fact of Copyright protection in respect of computer software and to promote a Private Members Bill to amend the Copyright Act so that such protection is self-evident to eveybody. You will, no doubt, receive press releases from the British Computer Society and ourselves regarding this matter in the near future.

Meanwhile, the CRA on their own are seeking advice as to whether or not it is legal to form a prosecution fund to finance a plaintiff in a High Court case, so that the few lingering doubts as to the applicability of Copyright to software, shall be dispelled forever.

In addition, the CRA Executive Committee will be informing members that their services as possible expert witnesses in any litigation concerning Copyright will be available without charge.

The CRA would also like to take this opportunity of complaining about the totally biased and one-sided view-point aired by the BBC program Newsnight. This was broadcast on Saturday the 28th January, and also we believe on a previous occasion in December. The impression that the BBC left with the viewer was that there is some grave doubt as to the applicability of the Copyright Act to computer software. This is totally untrue and, of course, was the subject of a previous press release from the CRA. In addition, the BBC was totally erroneous in giving the impression that software houses do not prosecute infringers of copyright. My company itself has been involved in one case, indeed it was the first case, and there have been several since. The only small amount of truth in the BBC's broadcast in this area was that no software house has **yet** taken the matter to an Appeal Court.

There have been at least six or seven cases in which an Anton Pillar order was obtained. In other words, in which a High Court judge agreed with the plaintiff that the Copyright Act does cover computer software.

It seems to the CRA that the BBC would be well advised, as a public body, to direct its attention towards obtaining the truth of a matter, rather than to exploiting rather dramatic interludes with the apparent sole purpose of obtaining a wider audience for its programme.

If, as it would seem, the BBC wish to broadcast dramatic incidents then they would do both the microcomputer industry, and the public as a whole, a far greater service by reconstructing the effect of a raid resulting from a High Court Anton Pillar order, instead of rather ineffectually hiding their cameraman away in a little corner in an attempt to catch an alleged offender. In other words, to make the subject matter of their program affirmative on the side of publicising the fact that offenders are in the wrong, rather than attempting to gain dramatic effect in emphasising their incorrect view that copyright does not apply to computer software.

The CRA would like to go on record in stating that it is unfortunate that the BBC apparently did not seek the advice of experts on the subject or, if they did, that they did not give the views of those people at least equal time with the side of the matter which appears to be far more attractive to them, namely the negative side.

It seems strange to the CRA that the BBC should quote Mrs. A. Staines. Firstly, she most certainly does not speak for the British Computer Society Copyright Committee as was implied by the caption on the screen. Moreover, if the views expressed in the transmission are the result of an interview with her which has not been cut or otherwise altered, then the views expressed are, to say the least, unusual and in particular, in our experience of sitting on the Committee, such views are not held by anybody else on that Committee. It is no secret that it is the unanimous (apparently with the exception of Mrs. Staines) view of the Copyright Committee and its legal advisers that the Copyright Act, as it is at the present time, most certainly applies to microcomputer software.

In summary, we feel that the BBC has decided that the subject of piracy would make good transmittable material and furthermore that the matter is far more dramatic when viewed from the negative point of view. The programme was harmful to the industry and represented not only bad reporting, but also a distortion of the truth.

## DYSLEXICS HELPED BY AN ACRON

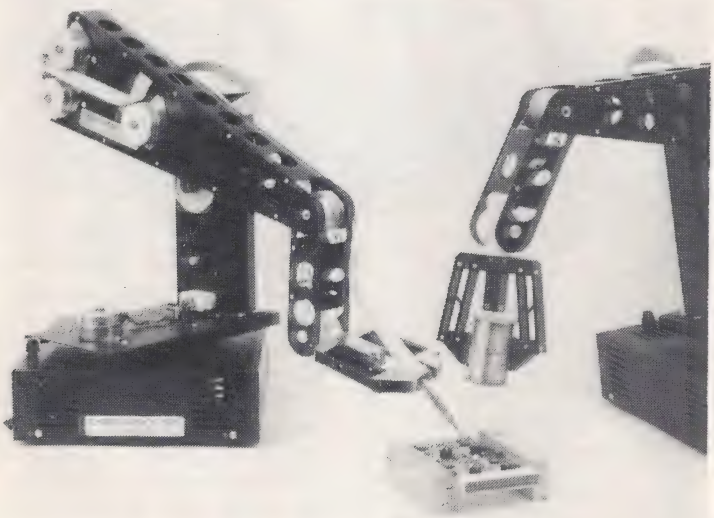
Duncan Goodhew — Olympic swimmer and one of Britain's best known dyslexics — visited the Dyslexia Institute's national headquarters at Staines on February 6, to accept an Acorn microcomputer from Bill Fulton, managing director, Sony UK, Ian Crammond, chairman and managing director of Teletape Video Ltd and Peter Thompson, chairman and managing director, Peter Thompson Associates.

The trio were representing Vision charity — an organisation set up by the video industy to help visually handicapped children — which has donated the computer to the Institute.

Also present were Dr Harry Chasty, the institute's director of studies and senior teacher Wendy Norton as well as some of the children who will be using the Acorn as a simple teaching aid to develop basic reading, writing and spelling techniques. With this computer, carefully controlled activities will be programmed to enable successive groups of children to develop these basic skills at their own pace.

Dyslexic children react very positively to information on screen, and at a later stage in their education could become very skilled in the use of computers. It is planned that this new computer will eventually become an important thinking tool for the Institute's students as they progress towards higher language skills, requiring them to process, store and retrieve information.

We don't know about you, but at Computing Toady we think this is a lot more worthwhile than Space Invaders.



## HERE COME THE CYBER MEN

The Hi-Tech company Cyber Robotics Ltd of Cambridge is dedicated to the development of a range of cost effective robots for use in a wide variety of applications. The Robot's operating system is an extension of FORTH.

The adaptability of FORTH permits speed control, which is important to keep attention at demonstrations. Likewise the language will allow slow movements when close analysis of critical control is required. Also, joints can be moved simultaneously, emulating a full size industrial robot and providing operators with the most realistic educational setting.

The Cyber 310 is compact, light and is manufactured to a high engineering standard which is particularly important to cope with the wear and tear inevitable in any educational environment. It has five degrees of movement, a programmable gripper action, plus a unique ability to rotate the shoulder 300 degrees in the vertical plane to operate on the opposite side with the arm upside down, therefore increasing the robot's flexibility.

Cyber Robotics are designing a complete package of software, changing the whole perspective of educational robots from the mere application of computer to robot to a complete educational programme. Very much related to this, Cyber Robotics Ltd are developing a "user club" which will keep all buyers and users informed of new application software available. Further information from Cyber Robotics Ltd, 61 Dutton Walk, Cambridge, CB5 8QD (telephone 0223 210675: telex 817662).

## LONDON COMPUTER FAIR

The Fifth London Computer Fair is to be held on Thursday 19, Saturday 21 and Easter Monday 23 April 1984 at the Central Hall, Storeys Gate, Westminster. After four successful years the Association of London Computer Club's London Computer Fair is on course to repeat the formula which has worked so well. The slot used this year has been successfully used by another hobby group for many years and being over the holiday weekend, helps the amateurs who keep the show on the road as officials, cashiers, stewards and all the many other jobs that they cover.

This year there is a Micro-Robotics Conference and

Demonstration presented by the ACC, the finals and the presentation of the winners of The Times Software Competition, a Networks and Communications Feature, presented by the ACC Comms Group on Saturday, the Bring-and-Buy Feature and on Easter Monday an auction of items you will find it hard to resist.

As in previous years the Fair is aimed at the hobbyist, home user, educational user and the small business user. It is also hoped to bring a great variety of extended computer activities to the general public and the enthusiast alike. To backup the features and demonstrations there are dealers in new, secondhand and surplus equipment, software suppliers, peripherals (printers, modems, buffers, and so on), publishers, book dealers and popular magazines.

The ACC clubs database will give enquiring visitors the location of nearby clubs and club officers will be able to compare notes. Up to the minute details will be available on Prestel (available in many public libraries) and by radio and press coverage.

At least 10 clubs will have stands showing the wide range of activities from games software to community projects for the disabled. The ClubSpot feature on Prestel will be on demonstration and updates can be observed. Watch the special offers go on line! All the clubstands will have networking demonstrations and for some 'hands on experience' as well as much more!

Club enquiries to Frank Spilsbury (01-303 8849) and Len Stuart (01-337 3747). Commercial enquiries to Tim Collins, CMP (01-930 1612).

## DATAPEN FOR THE 64

The Hampshire based company Datapen Microtechnology Limited has jut announced full details of its CBM-64 compatible lightpen and programs. Previously, dealers who were advance ordering the CBM-64 version had only seen preliminary details of the software to be supplied with each lightpen and had placed their requirements with the confidence gained from selling Datapen's BBC B, Dragon 32 and VIC-20 compatible lightpens.

There are three programs provided with each lightpen and Datapen's philosophy of providing the introduction software on tape as well as a printed listing, enables the user to access the lightpen's features (X-Y position, LED lamp and control switch) and use them easily in his own programs. The introduction program on the CBM-64 version also shows how to move and place sprites with the lightpen.

The second program, called Colour-Draw, allows the user to produce a drawing by transferring CBM's colour graphics characters from an on-screen menu to the required position using the lightpen.

The third program, illustrated in the photograph and called Hi-Res Draw, provides the user with the ability to produce a drawing in 200 by 320 pixel resolution directly with the lightpen and provides features such as line-drawing and turtle drawing.

A Sprite Creator and Editor program for use with the lightpen will be available shortly.

Datapen are now shipping the CBM-64 version in quantity to dealers and also directly to home users at £25.00 each. Further details may be obtained from Datapen Microtechnology Limited, Kingsclere Road, Overton, Hants RG25 3JB (phone 0256 770488).

## SPECTRUM/QL MONITORS

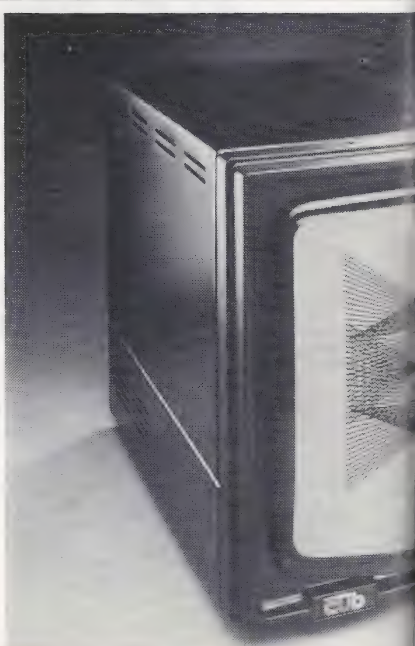
Microvitec have just launched their Sinclair Spectrum-compatible colour monitor on the consumer market. Sales in the education sector have prompted the company to make the monitor available to the many home users of the Spectrum — Britain's most popular home computer.

What the makers claim to be the only low complexity colour display equipped to handle Spectrum outputs, the 1431/MZ comes in a metal cabinet with die-cast frame surround finished in matt black to match the appearance of the Spectrum. Inside, the well-proven picture tube and control circuitry of the RGB/TTL input models is retained while an additional card carries the Spectrum interface which effectively converts to the RGB/TTL format the luminance ('U') and chrominance ('V' and 'Y') signals appearing at the output port of later Spectrum models. Since the interface can be switched in or out of the circuit, the monitor can also be driven by computers with conventional RGB/TTL outputs.

Designed specifically with the new Sinclair QL Computer in mind, the Microvitec CUB 1451/MQ3 features a 14" high-contrast, self-converging P.I.L. tube with a resolution of 653 addressable pixels horizontally and 585 vertically. These features, together with an 18 MHz bandwidth, make the new monitor ideally suited to the QL's high resolution graphics and 80-column text capability. Microvitec's drive circuitry ensures maximum reliability combined with a low power consumption figure (typically 65 watts).

The CUB 1451/MQ3 is the latest model to emerge from Microvitec's on-going product development programme. Like its Spectrum-compatible stablemate, the CUB 1451/MQ3 comes complete with dedicated cable and is housed in a metal cabinet with die-cast frame surround finished in matt black to match the appearance of the new Sinclair machine. All cable entries and controls are accommodated at the rear of the cabinet which, from the front, presents the classic look of current Microvitec CUB colour monitors.

Further details of this and other Microvitec products, including switch mode power supplies, are available from the Sales Department, Microvitec Limited, Futures Way, Bolling Road, Bradford BD4 7TU, West Yorkshire (telephone 0274 390011).



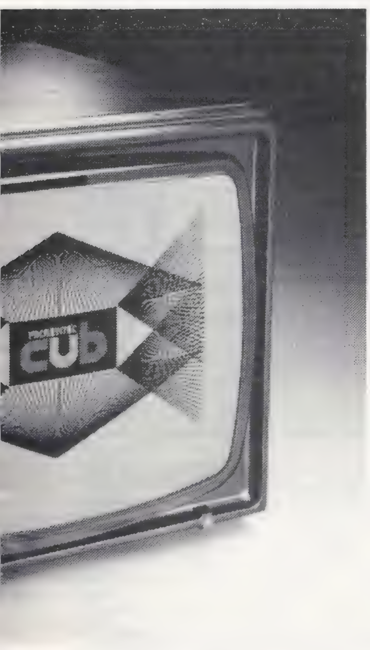
## LOCK UP YOUR FLOPPIES

The U.K. 92 Data Safe is a new concept in floppy disc storage and handling. Made from glass re-inforced polyester, the unique design combines high-impact strength and portability with security and fire resistance. The flame retardancy conforms to UKOVO:BS476 Part 7, Class 2, as approved by leading insurance companies.

The high-impact strength and the sturdy lock make Data Safe an ideal carrying case for up to thirty 5¼" floppy discs. Invaluable to all computer users, software houses, banks, accountants, couriers and so on, the built-in handle enables Data Safe to be carried easily, so that important records can be stored conveniently in a separate place, taken home at weekends and so on. It is available from Costerwise Ltd, 16 Rabbit Row, London W8 (telephone 01-221 0666) and all leading distributors for around £30.00.

## MZ-700 GETS DISCS

The Sharp MZ-700 microcomputer with 64K RAM, colour and sound at £249.95 has already proven popular as a home and small business machine. The full potential of this powerful, versatile micro is now realisable with the advent of an efficient disc drive unit. Kuma Computers Ltd, the major supplier of MS-700 software and hardware products are now the exclusive UK distributors of the SFD 700.




The SFD 700 is a stand-alone single floppy station which will connect directly with the MZ-700 I/O bus. It is supplied complete with interface card, cable, DOS and disc BASIC: no extra items are necessary. The station is built in an aluminium case which is coloured in ivory and brown.

The capacity of the disc drive is 280K, and the controller card will handle up to four disc drives. The disc BASIC is fully compatible with the S-BASIC of the MZ-700, but extra features have been added. The disc BASIC supports the 64K RAM of the MZ-700 and the Printer/Plotter, and it has additional monitor features to those built into the MZ-700.

The SFD unit is priced at £495.00 plus VAT. Further details can be obtained from Kuma Computers Ltd, 12 Horseshoe Park, Pangbourne, Berks (telephone 07357 4335).

## LYNX RUNS FASTER

Lynx owners can now add to the machine, up to four 5¼" disc drives, each with 200K capacity (formatted). Each drive is in a sleek, but rugged metal case with its own illuminated on/off switch and disc access light. Connection is via the plug-in disc interface, with a simple keyed cable connector provided with the interface module. Additional drives simply daisy chain on to the first (master) drive.

Equally important is the operating system. Lynx DOS is fully functional and available now. It adds 19 commands as an enhancement to the existing BASIC, enabling all needed functions for disc operation to be carried out.

Drives, interface and software are available ex stock for orders placed now. Contact Camputers Ltd, 33A Bridge Street, Cambridge CB2 1UW (telephone 0223 315063: telex 817207).

## DIAL-DATA COMMUNICATION FOR MICRO-WRITER

Communication of text and data between remote locations is now available with the Dial-data unit, distributed in the north-east of England by sole authorised distributor J.M. Office Supplies. J.M. Supplies is an appointed Microwriting Centre and director Mr Mulcahy is certain that: "The Dial-data unit will prove particularly useful for my Microwriter customers."

Text entered on the hand-held, six-keyed, Microwriter wordprocessor is transferred directly to the Dial-data Unit through its standard RS232 connector port. The single-line display of the Unit can be utilised for checking information before transmission. The Dial-data unit is then linked to an acoustic coupler so that text is simply downloaded through normal telephone lines to a receiving unit.

Recipients can see immediately what has been sent by scrolling the text through their Dial-data unit screen. The unit has its own 16K storage capacity so that messages can be held in the unit until it is convenient to deal with them. If a printed version of the material is required, the Dial-data unit can be linked to electronic typewriters or ordinary office printers which have RS232 interface ports.

The Dial-data unit will also communicate with microcomputers, wordprocessors and, of course, Microwriters, so that text can be transferred to any of them for storage and future reference. Incorporating an error-checking device, the unit will continuously try to get through to the intended recipient for 30 seconds but if there is either a fault on the line or it can't get through, it simply re-dials automatically and tries again.

In the near future, J. M. Offices Supplies will also stock another version of the all-British Dial-data System with direct modem links, which will not require acoustic couplers.

A major advantage of communication by Microwriter and Dial-data unit is the complete confidentiality it offers. The users can compose their own text on the Microwriter and then control the distribution of this material by sending it directly from their Dial-data unit to a specified recipient. This facility will be even further enhanced in the Spring of '84 when a 'scrambling' unit will be available.

The Dial-data unit, developed by Duplex Midlands Limited, is not only a very useful communications aid for Microwriter users, it will also provide communication between other RS232-equipped machines, such as electronic typewriters, as well as for Telex to Telex transcription.

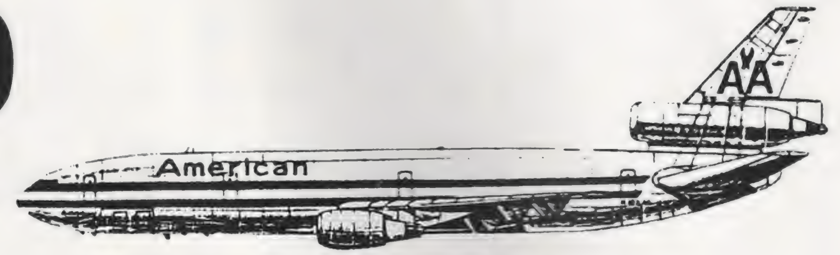
Cost of the Dial-data Unit, including accoustic coupler, is £650. The unit on its own is available at £495. A Microwriter costs £299, and is available from any Microwriting Centre nationwide.

# Software News

## INNOVATIVE
## TRS 80-GENIE SOFTWARE

***from the professionals***

# DC~10




This DC-10 program continues the line of highly accurate flying simulations stocked by Molimerx. Apart from the original Columbia Shuttle simulation, there are now programs to simulate the piloting of a 747, the Concorde, Airbus, and now the McDonnell Douglas DC-10.

The DC-10 is a three engine, wide bodied jet manufactured by McDonnell Douglas of the U.S.A. It has had rather an unfortunate history in service, in that there were problems with a cargo door and then an engine mounting which apparently was damaged in a maintenance service. Regardless of these problems, which may well have sunk the saleability of another aircraft, the DC-10 is still an extremely popular aircraft with the airlines. As will be seen from the illustration, it is quite easily confused with the Lockheed L1011 but it is, in fact, an entirely different aircraft.

As with all other major aircraft, there have been many versions of the DC-10. The first five development aircraft were actually started as long ago as January 1969. Rolls Royce got itself into troubles, and so McDonnell Douglas chose the General Electric CF6 turbo fan engine. It will be recalled that Laker Airways invested heavily in the series 10 version of the DC-10. However, taking the first series for descriptive purposes, the three engines developed 41,000 lbs. of thrust and the aircraft had a wing span of 155ft. 4ins. The maximum cruising speed was 584 miles per hour and the service ceiling was 35,000 feet. The range of 2705 miles was with a maximum payload of just over 100,000 lbs.

This program is a simulation of DC-10 flights over and around Europe. Both Gatwick and Heathrow airports are included, as are four on the Continent. As in real life navigation is by radio aids. DC-10 features no less than ten VOR beacons in the United Kingdom, and fourteen on the Continent. For the first time in our simulations Non Directional Radio beacons are included, four in the United Kingdom and seven on the Continent. Instrument Landing Systems and Distance Measuring Equipment are provided at all six of the runways upon which you can land DC-10. As with earlier simulations, wind both on the ground and aloft is included, as is a random engine out emergency. The instruments are as follows:

1. Indicated air speed gauge
2. Artificial horizon
3. Power setting for No.1 engine
4. Power setting for No.2 engine
5. Power setting for No.3 engine
6. Slat setting
7. Flap setting
8. Compass
9. VOR tracking instrument
10. Instrument Landing System
11. Clock
12. All up weight
13. Fuel
14. Fuel flow
15. Runway to go (on take off)
16. Vertical speed indicator
17. MACH speed
18. Precise pitch
19. Precise roll
20. Altimeter
21. Landing gear status
22. Nose wheel status
23. Wheel brakes status
24. Air brakes status
25. True air speed
26. Wind direction and velocity
27. Ground speed
28. Destination runway, place & number
29. Distance to go
30. Precise heading
31. Precise track
32. Data from No.1 DME/VOR
33. Data from No.2 DME/VOR

DC-10 is supplied with a free program which enables the user to calculate Flight Plans on his computer, to be used in DC-10.

An extensive illustrated manual is supplied. It takes the reader through the control panel in general and then in detail. Discusses the controls at length; general discussions are held on flying technique of DC-10 and then simple flight manoeuvres are described, such as normal take off, noise abatement take off, take off with engine failure, climb, cruise, turning, descent, approach, final approach and landing. Procedures in overshoots and engine out emergencies in various situations are described. Simple flight briefings, in other words, instructions for suggested flights, are also included. There are a number of Appendices, including detailed discussions of the VOR/DME navigation system and the ILS approach system. Purchasers of DC-10 may also buy the educational section of the Jumbo manual for £1 if they wish.

The program is compiled Basic and is disk orientated only. The compilation enabled the author to include very precise slow down loops in the source code. Thus, as we have said, the simulation is as exact as it is possible to get. The compatibility of the program with various disk operating systems and machines will depend upon the compatibility of those DOS's and machines with the Microsoft Compiler. As far as we know, on TRSDOS it is compatible on all Genie machines (with the exception of the Model III) and of course the Tandy Model I. It is also compatible with LDOS on these machines. On the Tandy Model III the choice of DOS is somewhat more crucial as many disk operating systems running on that machine are not compatible. TRSDOS 1.3, for instance, is not. A patched version of the run time file, to enable use on the Model III under LDOS or smal-LDOS is included in the package.

This DC-10 simulation will be released for the IBM PC machine mid-March. At the time this advertisement goes to press we are not sure whether the program will need to be compiled or not.

DC-10 (disk) ... ... ... **£19.55**
VAT inclusive, 75p P & P

**TEL: [0424] 220391/223636**

**MOLIMERX™ LTD**
**A J HARDING (MOLIMERX)**

**TELEX 86736 SOTEX G**

**1 BUCKHURST ROAD, TOWN HALL SQUARE, BEXHILL-ON-SEA, EAST SUSSEX.**

TRS-80 & VIDEO GENIE SOFTWARE CATALOGUE £2.00 plus £1.00 postage (postage refundable on order).


# MTX MAGIC

Owen Bishop

**Once upon a time there was a company called Memotech who made ZX81 peripherals. Then one day they decided to make their own computer, with sprites at the bottom of the garden and NODDY, too. Will they live happily ever after?**

The MTX-500 is full of surprises and, let it be said from the start, the surprises have mostly been pleasant ones. The first surprise came when the MTX-500 was first connected to the TV set and switched on. The 'Starting Out' section of the provisional Operator's Manual (253 pages, A4 size) had not mentioned whether or not the machine had a built-in loudspeaker or whether it sent its sound signals to the TV. The obvious test, assuming that it would understand ASCII, was to type 'Control-G'. This is the so-called BEL code which originally caused a bell to ring on a teletype machine, but which nowadays makes most computers emit a beep. Surprise number 1 — a clear, bell-like 'ding' from the loudspeaker of the TV set! Obviously the sound generator of this machine is above average in its capabilities.

## MODEL LOOKS

But this introduction is running a little too far ahead of events. Let us go back to the stage when the MTX-500 was unpacked from its carton. What we saw was no surprise, for the advertisements had already shown us what the machine should look like. Perhaps it might count as a surprise to find that it actually does look as smooth and sleek as it appears in the photogaphs. It has the 'long, low look', with a satin finish in anodised black aluminium. Its appearance will give it pride of place, either in the living room or on the executive's desk. The keyboard measures 48.5 cms across, which is considerably greater than most popular micros.

The reason for this is that the keys are set out in three areas, with a comfortable margin around each. There is the keyboard area proper, with 59 keys. These include the usual alphanumeric keys and symbols, all with auto-repeat. The control keys are Escape, Control, Alpha(betic), Lock, Shift (two keys), Back Space, Line Feed and Return (which is rather too small). These keys are placed in their conventional positions, so there was no trouble adapting to their layout. There are also two unmarked keys on either side of the space bar — these are the Reset keys. As a safety factor, both of these keys must be pressed together to reset the machine.

To the right of the main area is a key-pad of 12 keys. These have several functions, depending on the current mode of operation. Eleven of them are set out for entering numeric data, duplicating the action of the numeric keys in the top row of the main keyboard. They are arranged in a conventional 'calculator' format, and include a decimal point key. The keys are also marked for other functions, including four direction keys useful for games players, but also used for directing the cursor around the screen.

Several other of these keys are used for editing, including a Delete key and an Insert key. There is a Page key which toggles the machine into one of two text-entry modes. In Scroll Mode, entering new text on the bottom line of the screen causes the screen to scroll upward. In the Page mode the cursor jumps from the bottom right of the screen to top left; new text appears at the top of the screen, replacing that which is already there. Other keys in this area are EOL, which deletes all characters from the cursor as far as the end of the line, CLS which clears the screen, Home which takes the cursor to the top left of the screen without clearing it, and BRK, which breaks a running program.

There is a block of eight function keys (F1-F8) to the extreme right of the keyboard. These are not programmable keys in the sense of those found on the BBC Micro, for example. Like the function keys of the Commodore 64, they can be used as special control keys in programs. They return ASCII codes when pressed, giving 16 possible codes in all (shifted and unshifted).

## IT'S SPRING AGAIN

All the keys are lightly sprung, yet return rapidly to their position. We were reminded of the keyboard of the TRS-80 Model I. I have never found a keyboard on which I, a non-typist, could type as fast and with so few errors as that one.

Perhaps and greatest surprise on removing the MTX-500 from

its packing was not the appearance but its weight. Robustly constructed, it sits firmly on table or desk. The metal case, with its black surface, is a guarantee that the circuitry is unlikely to become overheated, even after long periods of use. The fact that the power pack is a separate unit (there would be no room for a sufficiently powerful transformer in a case only 6 cm deep at its deepest) is a further safeguard against overheating. The leads of the power pack total just over 3 metres in length, allowing you to operate the computer at a reasonable distance from the mains socket. The power pack is sturdily made; it has an on-off switch which incorporates a pilot light.


## INPUT/OUTPUT

The cartridge port is situated at the left-hand end of the case. The aperture reveals a 30-way connector giving access to the Z80 address bus, the data bus, the data bus and all the usual Z80 control lines, as well as power supply lines and various other connections. This connector may be used for plug-in cartridges, or for connecting a variety of other peripheral devices.

Full details of the connector are given in the Operator's manual, which also includes essential circuit diagrams and technical details of the main I/O chips. A clip-on cover is provided for the cartridge port, so that it can be neatly closed if it is not required.

All the other connections are on the rear of the case. Another surprise came when we started to plug in the power and TV leads. With most micros one has to twist the case at peculiar angles to see where each lead is to be plugged in. The rear of the MTX's case is rounded and the legends for each I/O aperture are marked above the cutouts. Thus they can be clearly seen when looking vertically down on the machine from the front. This makes it easy to plug in the connections correctly, without having to perform contortions. The connections comprise:

- Two RS-232 ports (communications board required)
- Monitor output
- Hi-fi output
- Power input (from power pack)
- TV output (UHF)
- Parallel I/O port (internal socket)
- Parallel printer I/O (Centronics)
- Cassette I/O
- Two joystick inputs.

There is no output for an RGB monitor: however, we have found that the 16 colours (see Table 1) that the MTX produces all show up crisply and clearly on our TV set. Whereas on some computers with 16 colours there are several combinations of screen and background colour which blur into indecipherability, there are very few colour combinations on the MTX that suffer from this defect. Incidentally, the colours are all available all of the time and, since the MTX has a separate 16K video memory, you do not suffer the penalty of losing user memory by using all colours freely.

## TABLE 1

**MTX colours.**

| | |
|---|---|
| 0 | transparent |
| 1 | black |
| 2 | medium green |
| 3 | light green |
| 4 | dark blue |
| 5 | light blue |
| 6 | dark red |
| 7 | cyan |
| 8 | medium red |
| 9 | light red |
| 10 | dark yellow |
| 11 | light yellow |
| 12 | dark green |
| 14 | grey |
| 15 | white |

The cassette port has only Mic and Ear connections, no provision being made for controlling the motor of the recorder.

## THE MANUAL

The machine supplied for review was a pre-launch model and its manual was a provisional one. In spite of this it contained a mass of information and remarkably few errors. The erros were mainly typing ones, not errors of fact, and these, we are informed, have been corrected in the final version (this is now ready). There was no index, but a comprehensive contents page tells you where the main summary tables may be found.

The manual begins, in a computerish way, with Part 0, the introduction. This contains a description of the machine and its keyboard. It also includes an overview of the MTX system, with special reference to its languages. The computer holds four languages in its 24 kilobytes of ROM. The main language is MTX BASIC. This has much in common with other BASICs, there being a strong resemblance to Sinclair BASIC, so ZX81 and Spectrum owners will rapidly feel at home. However, as we will describe later, it has many features of its own. The manual includes a clear and concise introductory course in BASIC, with examples and exercises.

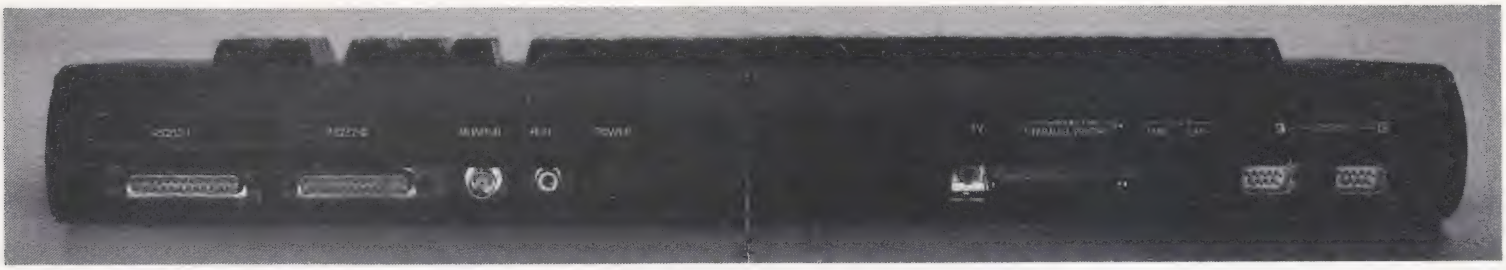
NODDY is a language specially designed for handling text. It has few commands, is easy to learn and use, and may be called from BASIC programs. If you want your programs to display screenfuls of text, or to allow the user to key in screenfuls of text, having NODDY around is a great help.

A page (screenful) is formatted simply by typing it directly on the screen. The direction keys allow you to move the cursor around and place text exactly where you want it with the minimum of fuss. The page is recalled instantly with exactly the same format, simply by typing its name, or in various other ways by using the NODDY commands. It is much simpler to handle text with NODDY than to get involved with masses of PRINT or INPUT statements in a BASIC program. Or, if you require a text-handling program such as an address book, it can be programmed entirely in NODDY.

The third language in the ROM is the MTX Graphics package. This contains a comprehensive set of graphics commands, with a strong flavour of Logo. More about this later.

## ASSEMBLER

Finally there is the Z80 assembler. This allows whole programs to be written in machine code. It is particularly useful to the average programmer in that sections of programs



or even very short routines may be written in assembler and automatically inserted into a BASIC program. This means that you can speed up the time-consuming parts of a games program, for example, making the computer work that much faster where it really has to. The rest of the program can be written in nice, easy BASIC.

The manual does not set out to explain machine code or assembler mnemonics, but assumes you already know about these things. The detailed instructions given for the assembler itself are clearly set out.

## SAMPLE PROGRAMS

Before diving into MTX BASIC, we decided to try out the sample programs which came with the machine. There were five cassettes altogether, Memotech having thoughtfully provided a blank C-15 cassette and a head-cleaner tape along with the demonstration tape and the two games. The demonstration tape loaded very quickly, first time, as did the games tapes. During loading, the signal from the cassette recorder is heard from the TV loudspeaker. This is a sure indication that something is happening, but the sound is rather irritating. Since the sound is coming from the TV and not from a speaker built in to the computer, it is no trouble to turn off the sound while loading.

The demonstration tape shows off many features of the MTX-500, including its Logo graphics, its sprites and its impressive sound generator. There are also several screenfuls of text, demonstrating NODDY at work.

The two games cassettes included were both by Continental Software, who are producing an increasing range of MTX software. The Toado tape displayed the 16-colour high-resolution graphics of the MTX to good advantage. It is a fast machine code game based on the popular 'amphibian with desires to negotiate road and river' motif, and is well up to standard. The MTX draughts has 10 levels of play — I tried it at level 4, which brought rapid and unerring responses from the computer. In spite of all my cunning ploys, it beat me soundly.


## MTX BASIC

This has a useful range of statements (Table 2). INK and PAPER are used to set the foreground and background colours and will be familiar to Spectrum users. LET is obligatory in assignment statements, another Spectrum-like feature. As might be expected, the command NODDY calls NODDY. PLOD is the word used to 'RUN' a program written using NODDY (hang on, this is getting a bit silly! Ed). CSR is used to position the cursor anywhere on the screen. It has the same action as TAB(X,Y) in other BASICs, though CSR seems a more sensible name.

MTX BASIC lacks procedures and user-defined functions.

PANEL is one of the words associated with the use of the assembler. Its function is to switch on the front panel — in this mode the screen displays the contents of the Z80's registers and a portion of RAM or ROM. In this mode there are 16 single-key commands which allow the user to roam through memory, examining it or altering it (if it's RAM!), or having it disassembled to mnemonics or displayed in its ASCII equivalent. These and other functions make this a valuable aid to the machine-code programmer.

CRVS and VS are words for creating and enabling virtual screens. These are text or graphics windows — up to eight of them may be created, in any size or shape, using CRVS. The monitor itself uses some of these: a 'List' screen of 19 lines, starting from the top of the screen; an 'Editor' screen consisting of four lines, which behave as one (you type your BASIC lines into this); and a 'Message' screen, the bottom screen line.

When you type in a line of BASIC program it appears in the 'Editor' line, near the bottom of the screen. When you press Return, it is transferred up into the 'List' screen, which scrolls upward as successive lines are added. Each program line is checked for syntax before it is accepted into the 'List' screen and, if there is an error, this is reported in the 'Message' screen.

MTX BASIC has a large number of words associated with its sound and graphics facilities — we will deal with these later.

The BASIC allows long variable names, up to 150 characters long. This limit is set by the amount of space available in the 'Edit' screen. Words of this length are distinguished from each other by the MTX — all the letters are significant. You can type in variables and BASIC keywords in lower case, but these are converted to upper case when the line is accepted into the 'List' screen. Thus no lower case letters appear in variable names. Lower case letters between quotes are, of course, not affected.

## BENCHMARKS

Before leaving the subject of the BASIC we must look at the results of the Benchmark tests (see Table 3). The listings used were the standard benchmarks published in a previous issue of **Computing Today** and in the Winter'83 edition of **Micro Choice.** The latter issue

quotes tests made in CT on the Spectrum, Dragon 32, Commodore 64, NewBrain, Osborne 1, Sirius 1 and the BBC Micro. In all the tests the BBC machine stood out from the rest. Our tests showed the MTX to be average at BM1, and well above average (ie faster) at BM2 to BM7. In BM4, BM5 and BM6, it was beaten only by the BBC Micro. We were therefore surprised when it did badly on the final test (BM8). It took three to four times as long as the other machines. This benchmark is the one which tests the arithmetic functions: exponentiation, log and sine. Dropping each of these from the program in turn, we found it was reasonably quick at exponentiation, but very slow at logarithms. Actually, we had to use LN in this listing instead of the LOG specified in the benchmark, since LOG is not available on the MTX. The conclusion is that the MTX is not a machine for those who want to perform elaborate and repetitive calculations involving logarithmic and trigonometric functions.

## ALL'S UNFAIR . . .

Benchmarks are notoriously unfair in that they can test only those commonplace features possessed by most BASICs, and cannot reveal the merits of any special features of a given machine. Most users will want trig functions not for mathematical calculations as such, but for plotting circles, calculating the orbits of spacecraft and so on. These calcuations are already provided for by BASIC words such as CIRCLE and ARC, ANGLE and PHI. The CIRCLE statement produced a circle more quickly than did a circle-drawing BASIC procedure running on the BBC Micro, and using SIN and COS. So for the majority of users, slow calcuation of the logarithmic and trigonometric functions is immaterial.

## SOUND

There are four sound channels, three of which produce tones while the fourth produces pink noise. These have a range of about 10 octaves. The sound channels can operate simultaneously. Sound commands can be stored in a sound buffer, so the computer can get on with other tasks while sound effects are being produced.

The SOUND command can have either three or seven parameters. With three parameters, a sound is continuous until it is stopped. The parameter which controls pitch can take any value in the range 0 to 1023. With such a large range, it is easy to produce a completely smooth change in pitch from one extreme to the other. There is no tendency for an ascending or descending note to sound like a scale being played. With seven parameters after the keyword, a wide range of more complex sounds can be generated — the frequency and volume can be altered at chosen rates during the sound. This can be a complicated matter, as might be expected, but the operator's manual deals with sound generation in great detail. Its example of the sound of spacecraft blasting off and then ascending into space is both realistic and impressive.

## TABLE 2

**Command Words.**
**MTX BASIC**

| | | | |
|---|---|---|---|
| BAUD | ELSE | INPUT | ATTR |
| CLOCK | STEP | LIST | COLOUR |
| INK | CSR | LOAD | ADJSPR |
| PAPER | DIM | PRINT | MVSPR |
| EDIT | GOSUB | OUT | SPRITE |
| GOTO | LLIST | POKE | CTLSPR |
| IF | NEW | READ | NODE |
| LET | ON | SOUND | GENPAT |
| LPRINT | PANEL | PLOT | RANGLE |
| NEXT | RETURN | CODE | WINDOW |
| NODDY | SAVE | OFF | RESTORE |
| PLOD | DRAW | TO | SELECT |
| PAUSE | FKEY | REM | EDITOR |
| RAND | THEN | CLS | DSI |
| RUN | CONT | ASSEMBLE | AANGLE |
| STOP | CLEAR | AUTO | SNDBUF |
| VERIFY | DATA | VS | ARC |
| CIRCLE | FOR | CRVS | LINE |

**MTX Operands**

| | | | |
|---|---|---|---|
| + | / | > | <= |
| − | ↑ | < | < > |
| * | = | >= | |

**MTX Functions**

| | | | |
|---|---|---|---|
| AND | ASC | PI | SQR |
| ABS | RND | OR | USR |
| EXP | NOT | ATN | LEN |
| SGN | COS | LN | MOD |
| TAN | INT | SIN | |
| VAL | PEEK | INP | |

**MTX Strings**

| | | |
|---|---|---|
| CHR$ | RIGHT$ | TIME$ |
| LEFT$ | INKEY$ | GREAD$ |
| MID$ | STR$ | SPK$ |

## GRAPHICS

There are two aspects to graphics on the MTX. First, there are the commands associated with drawing lines and shapes on the screen. The commands CIRCLE, ARC, DRAW, LINE, PLOT have obvious functions. PHI and ANGLE control the orientation of drawing, so allowing complex shapes to be drawn with relatively few commands, and allowing these shapes to be rotated easily.

The second group of commands are associated with the sprites. The MTX has an excellent sprite capability — there may be up to 32 of them, in different colours, all moving independently on the screen. Each sprite has a priority with respect to the other sprites, so that one can be made to move in front of another. The sprites are generated by GENPAT (which is also used to produce user-defined characters). The sprites can have an eight-by-eight or 16-by-16 pixel design.

A sprite can be displayed on normal scale or magnified to double size: it can also be given its own motion, allowing it to cruise around the screen automatically, or it can be moved to specified positions. Its motion can be reversed, using a single command, as would happen when it bounces off a 'solid' object. There is the unusual effect that a sprite may be made to draw a line (in any colour), leaving a trail as it moves across the screen.

## SPRITELY COMMANDS

The set of commands provided allow the user to control the sprites with the minimum of programming effort. As might be expected, some of the commands have numerous parameters: however, there is a command, ADJSPR, which allows the programmer to alter just one parameter of a given sprite without having to re-state all the others. This not only makes programming easier, but allows the program to run more quickly.

The above are only a few of the things that can be done with the sprites produced by the MTX's Texas TMS9918 video display processor (same family as the chips in the TI99/4A and

## TABLE 3

| BM | Time (secs) |
|---|---|
| 1 | 1.4 |
| 2 | 4.7 |
| 3 | 11.0 |
| 4 | 10.8 |
| 5 | 12.6 |
| 6 | 22.4 |
| 7 | 39.3 |
| 8 | 42.7 |

Benchmarks.

Electronics Today International's Cortex computer project). Full technical details of this chip are given in the operator's manual.

## CHARACTER SET

As well as the normal character set, the MTX-500 provides fonts for other languages: American, English, French, German, Swedish and Spanish. The fonts include all the special letters and accents used in these languages, with, of course, a corresponding loss of certain of the mathematical and punctuation symbols. The foreign language fonts, together with the text-handling facilities of NODDY, could make this a good computer for language-teaching programs.

## SUMMING UP

The MTX-500 is a machine with a flavour of its own. It has great scope which, during the time we have had it, and in the limited space of this review, we have not been fully able to explore. Programming it is straightforward once you have learned the special MTX BASIC keywords. The assembler makes machine code programming easy for those who are familiar with the Z80 MPU. NODDY is a useful text-handling language, making the computer suitable for business applications. The excellent graphics and sound facilities make it a first-rate computer for the games enthusiast. Its large memory, expanded to 64K and beyond, and the fact that none of this is robbed to provide for the graphics, makes it likely that some extensive and elaborate games will be developed by this machine.

Overall, we liked its appearance, construction, facilities and action very much and consider it good value for money in comparison with other machines in this price range.

## FACTSHEET

| | |
|---|---|
| **CPU** | Z80A |
| **Clock** | 4 MHz |
| **ROM** | 24K |
| **RAM** | 32K, expandable |
| **Video RAM** | 16K |
| **Languages** | MTX-BASIC (including MTX graphics)<br>NODDY<br>Z80 Assembler |
| **Keyboard** | 79 keys, including keypad and eight function keys |
| **Display** | 24 lines of 40 characters on TV or monitor<br>Up to 16 colours<br>Up to 32 sprites<br>Eight user-definable virtual screens |
| **I/O** | Cassette port (up to 2400 baud)<br>Parallel I/O port<br>Joystick ports (2)<br>Hi-fi<br>Monitor<br>TV<br>Cartridge<br>Printer (Centronics) |
| **Options** | Communications board (two RS232 interfaces)<br>ROMs for FORTH, PASCAL<br>Colour 80-column board<br>Memory boards (32K, 64K, 128K, 256K, 512K)<br>Disc systems<br>Silicon disc (256K) |
| **Cost** | MTX-500 £275 (including 15% VAT) |
| **Supplier** | Memotech Ltd, Station Lane, Witney, Oxon OX8 6BX. Telephone: Witney (0993) 2977. |

# EASYCODE PART 2

**Simon N. Goodwin**

Now that we've presented the Easycode program for one machine and explained the basic principles, here's how to convert it for a variety of machines. We also have the first of our example routines.

In last month's article we introduced 'Easycode', a program which lets you learn machine code (almost!) as painlessly as if it were BASIC. If you missed that article you need a copy of the listing from a back-issue of the magazine. This month we explain how you can convert the program for other machines. We also press on with example programs, including arithmetic routines and even memory-mapped graphics!

## EXAMPLE PROGRAMS

Last month we experimented with some trivial Easycode programs. This issue we use most of the instructions, and explain how to crack one of the major problems of machine code programming — the storage of numbers larger than the computer can fit in a single location.

Program 1 performs a common task — it adds up the values in locations 50-59, and stores the total in location 99 (at the end of memory). We've changed the format of our Easycode listing so

### TABLE 1

| | | | |
|---|---|---|---|
| 1000 | CLEAR command | 11000 | RUN command |
| 3500 | Scan keys | 11500 | Fetch instruction |
| 4000 | Clear line | 12000 | SAVE command |
| 4500 | Accept number | 12500 | Accept file name |
| 5000 | Update registers | 13000 | LOAD command |
| 6000 | Position cursor | 14000 | HELP command |
| 8000 | Display number | 15000 | QUIT command |
| 9000 | Draw full screen | 16000 | STORE command |
| 10000 | Accept command | 16500 | Update memory |
| 10500 | STOPPED message | 20000 | Execute instruction |

**Easycode routines.**

that it closely resembles the format of a 'real' machine-code or assembler listing. The program is divided into four vertical columns. The first column (0, 2, 4, 5 etc) is the 'address' at which the instructions are stored. Program 1 occupies 16 locations, since it starts at address 0 and goes on to 15 (the JUMP at address 14 uses two locations).

The second column contains the data which should be stored in each location. The instruction code to load the 'A' register is 1, hence location 0 contains 1. The value to be loaded is 0, so location 1 contains 0 — and so on. At the end of the listing, the code 10 corresponds to a JUMP instruction. The 4 following it (in location 15) tells the computer where to JUMP to. The numbers in the second column (1, 0, 11, 20 etc) are the ones you enter using the STORE command.

Column three contains the mnemonic form of the instructions. A computer can't understand mnemonics directly — they are just shorthand for human beings. You can get programs — known as assemblers — to convert mnemonics into the corresponding numbers. The expanded version of Easycode contains a built-in assembler, as well as a disassembler which performs the reverse function, taking the numbers stored in memory and converting them into their mnemonic equivalents. The program lines needed to add these features to Easycode will be published in next month's CT.

The last column in the listing contains 'comments'. These are English phrases added by the programmer to make the purpose of the code more clear to someone reading the listing. Comments are similar to REMS in BASIC; they are ignored by the computer.

```
 0:   1  0    LOAD   A;0      Total so far is zero
 2:  11 20    LOAD   X;50     X 'points to' each value
 4:  19       ADD    A;@X     Add a value to the total
 5:  16 29    SUB    X;59     See if X has reached 29
 7:   9 12    JUMPNZ ;12      Jump if X-29 isn't zero
 9:   3 99    STORE  A;@99    Store the total at 99
11:   0       HALT            That's all, folks
12:  15 30    ADD    X;60     Point X at next value
14:  10 4     JUMP   ;4       Go back and fetch it
```

**Program 1. Adding up a list.**

## ADDING FOR BEGINNERS

The first two lines of Program 1 are straightforward. STORE the program and RUN it from address 0. You see the value 0 loaded into A, and 50 loaded into X. Both registers are being used for their designed purpose — the A register contains the running total and the X register is used as an index, 'pointing to' the address of items in the list. This is where the instructions using '@X' come in useful. They fetch, add or otherwise manipulate the contents of the memory location which is numbered in the X register.

If X contains 3, LOAD A:@X will fetch the contents of location 3. The instruction would be read 'Load A with the value at location X'. The third instruction adds the contents of the location pointed to by X. The X register is being used rather like a bookmark, keeping track of the place the computer has reached.

It's all very well having the X register marching through memory, pointing accusingly at everything it passes, but how do we tell when we have reached the end of the list? We know that the last item is at address 59, so we must stop once X reaches 59. Most processors have an instruction which 'compares' a register with a value. These instructions work by subtracting the required value and then setting the flags according to the result. In Easycode we have to use an explicit subtraction (wiping out the previous result in the process), but we shall see that this is not a problem.

### TABLE 2

| | |
|---|---|
| I,J,K | Miscellaneous integer counters etc. |
| ROW | Cursor vertical address 1 (top) - 16. |
| COLUMN | Cursor horizontal address 1-32 |
| T$,T1$,T2$ | General purpose strings |
| P | Program counter |
| ABRT | Non zero when program has been aborted |
| CARRY | Carry flag |
| ZERO | Zero flag |
| R(0) | Accumulator (register 0). |
| R(1) | Index register (register 1) |
| M(0) . . . M(99) | Memory array |

**Variable list**

To test for the value 59 we merely take 59 away from the contents of X, with a SUB X;59 instruction. If the result is not zero, X does not contain 59; we must go on to the instruction at location 12. If the result IS zero, Easycode ignores the JUMPNZ (jump if not zero) at location 7, and continues to the next instruction, at location 9. This stores the value in A 'at' location 99. The program halts when the '0' code at location 11 is encountered.

Meanwhile, if we're still trolling along the list, we've arrived at location 12 with a problem on our hands. Depending upon your

## CONVERTING EASYCODE

The program was deliberately written for easy conversion to run on other computers. As much as possible, machine-dependent routines to read the keyboard, position the cursor and so on have been collected in one place. 'Easycode' uses a subset of the BASIC language — it will run on almost any computer with BASIC, a TV display and the facility to handle strings of characters.

Table 1 lists the routines which make up the Easycode interpreter. More than half of the routines should run without changes on just about any BASIC computer. In this section of the article we will go through the routines one by one, explaining their purpose and giving examples of the conversions required. None of the REMS (which include comments introduced by an apostrophe) need be entered.

There are five points which should be observed throughout. The listing assumes that the program is running on a computer which handles floating-point (decimal) numbers (a TRS-80 or Genie). A few systems only recognise integers (whole numbers). The program will work on such machines, so long as they have the other required features, but INT expressions, used to round-down numbers, should be ignored; integer BASIC always rounds down. Lines marked 'F.P. BASIC only' may be omitted on integer-only systems.

The CLS command is used to clear the screen at various points. If your computer doesn't have a clear screen command, you can probably simulate it by printing a special 'clear' character or group of characters. Alternatively you can set up a 'CLS' subroutine which calls the 'clear line' routine (line 4000) for every line of the display.

The next three 'general' points are addressed mainly at users of Sinclair computers, although they may concern a few other users. The program uses the format IF expression THEN line number. This may have to be entered in the form IF expression THEN GOTO line number.

Easycode assumes that array subscripts start at zero. If the instruction PRINT R(0) gives an error message, you'll need to add one to the subscript of every array reference in the program. In retrospect, this stems from bad design — in the interests of portability the original program should have ignored the existence of the zero subscript.

Table 2 contains a list of the 14 variable names used. These all differ in the first two characters, and they do not contain BASIC words (hence ABRT not ABorT). If your computer doesn't allow strings to have names or more than one character you must rename T1$ and T2$. If necessary, enter a LET statement at the start of every line which presently begins with a variable-name.

## A MODEL PROGRAM

The first few lines of the program are used to set up an 'empty' computer model. They are consequently used whenever the program is first RUN, and after a CLEAR command, which works by starting the program again from scratch. Line 1000 will not be needed on most computers — it sets all variables to zero and reserves space for up to 100 characters of strings. The DIM statements in line 1010 must be altered, as described earlier, if your computer does not allow zero subscripts.

The SCAN KEYS subroutine will run exactly as listed on a Spectrum or Dragon. The INKEY$ function returns an empty string ("") if no key has been pressed; otherwise it returns the character corresponding to the key. Note that the end-of-line key is assumed to produce CHR$(13), and the space key is expected to give CHR$(32). The key feature (sorry) of this routine is that it should go on without waiting if no key is pressed. If you don't know how to do this on your computer, consider using joystick control (usually by a PEEK) instead of the keyboard.

The CLEAR LINE subroutine should not need changing. It simply positions the cursor, prints 32 spaces, and puts the cursor back at the start.

ACCEPT NUMBER, from line 4500 onwards assumes your computer uses the ASCII character set. If necessary replace the colon in line 4510 with the character which follows "9" in the sequence recognised by your machine. The action of line 4510 is to reject any entries which do not start with a digit. A simple greater-than-or-equal-to-"9" test is unsatisfactory since, by convention, the string "9" is greater than "9".

The UPDATE REGISTERS subroutine is straightforward.

POSITION CURSOR tells the computer that the nect character to be printed should appear on row ROW and column COLUMN, assuming that the top left position on the display is ROW 1, COLUMN 1. On a Dragon use PRINT @ COLUMN+ROW*32−33, "";. A BBC Micro will require PRINT TAB(COLUMN−1, ROW−1);. The Spectrum version is PRINT AT ROW−1,COLUMN+1;. Atari owners should use LOCATE COLUMN−1, ROW−1. if you've got a terminal which recognises the VT52 escape sequences, PRINT CHR$(27);"Y";CHR$(31+ROW); CHR$(31+COLUMN); should work. if the worst comes to the worst you may have to HOME the cursor to the top left corner and print 'down' and 'right' characters repeatedly.

Next we come to the DISPLAY NUMBER subroutine, which starts at line 8000. This prints exactly two characters which indicate the value of N, from 0 to 99. The only tricky thing here is making sure you output two characters (one of them a space) for values less than 10. If your computer allows PRINT USING, use it!

The DRAW FULL SCREEN routine contains only one potential pitfall — the multiple NEXT statement in line 9110. Some computers will require separate statements for each loop (NEXT J amd NEXT ROW).

The subsequent three routines, ACCEPT COMMAND, STOPPED and RUN COMMAND, should not need changing. FETCH INSTRUCTION contains a single odd statement — the ON . . . GOTO at line 11680. Space has been left for this to be written out in full:

```
IF I=1 THEN GOTO 20000
IF I=2 THEN GOTO 20100 etc.
```

But a computed GOTO would work as well:

```
GOTO 19900+(100★I+400★(I>16)
```

so long as an expression such as 2>1 prints as '1' on your system (if it returns '−1', replace the second plus in the computed GOTO with a minus). You may have to use a mixed approach if your BASIC won't allow long lines.

The SAVE and LOAD subroutines occupy lines 12000-13240. These are perhaps the most difficult part of the program to convert successfully. If in doubt, fix these last. The routines merely SAVE and LOAD the array M() under the name in T$. Most computers require that you OPEN and CLOSE the data file before and after manipulations.

On a Spectrum you can get by with just SAVE T$ DATA M() and LOAD T$ DATA M(). Put ★"m";1; before T$ to save and load to microdrive (through cassette is almost as quick!). Most other computers use PRINT and INPUT to access files. The only difference from normal (console) access is a 'channel number' which tells the computer to use the cassette or disc, rather than the display. On the Genie '−1' is the channel number and '£' (typed as a hash) identifies it.

The rest of the program should be easy to convert, since it uses very simple statements. If your computer doesn't recognise PRINT, IF, GOTO, GOSUB and assignments, you're in real trouble!

approach to arithmetic, you'll find the next explanation very simple or very devious — please bear with us in either case.

When we reach address 12, register X contains 59 less than the address of the place where we got our last value from; or 60 less than the address of the value we need next. At first sight this is a hard problem to sort out, because we must have taken 59 away from X when there was less than 59 in X to start with! We can't store negative numbers in Easycode, so what's the result?

You can guess the answer if you remember the way the Carry flag works. If the number goes over 99, Easycode sets the Carry flag — carrying 100, if you like — and leaves the remainder in the register. Hence 69 + 42 gives 11, carry one. So what is 50 — 59? The answer, according to Easycode and (in principle) every other micro machine code, is 91, borrow one. The sum is treated as 150 — 59. This may bring back memories of school long-division — if it does, sorry!

This rather arithmetical explanation has a point (in case you were wondering!). Since 'carry' and 'borrow' work exactly the the same way in machine-code, it follows that, whatever number you start with, you'll get the same number back if you subtract, and then add, any other number. This is obvious in normal arithmetic, where negative numbers are allowed, but it seems odd when you can say:

50 − 59 = 91
91 + 60 = 51

The ADD on line 12 has the effect of setting X to the value it had before the subtract, plus one. So we didn't lose the result during the subtract, after all.

Now X is pointing to the next value to be added to the total. We can go back to the ADD A;@X instruction, using a simple JUMP;4.

## CHEQUERED FLAGS

STORE and RUN the program until you're happy that it works. Put different values in locations 50 to 59 to test it. Hopefully you're not flagging (sorry) yet, because there's another problem to be solved. What happens if the total is more than 99?

We can't store such a total in the A register, or in any one memory location, since there's only room for values between 0 and 99. But what, I hear you whistling in the dark, is to stop us using two memory locations? After all, we can only represent values between 0 and 9 with a single digit, but that doesn't stop us sticking them together in clumps to make tax demands and such-like.

We can count up to 9999 if we use two locations, one for the 'hundreds' and one for the 'units'. Since 10 individual locations can only contain separate numbers that add up to a maximum of 990, two locations will be plenty. Program 2 solves the problem. Location 98 contains the hundreds and location 99 the units of the result. While the program runs, the units are stored in the A register. STORE and test this program too.

```
 0:   1   0   LOAD     A;0        Clear the total
 2:   3  98   STORE    A;@98      No hundreds yet
 4:  11  50   LOAD     X;50       X points to the list
 6:  19       ADD      A;@X       Add an item
 7:   8  19   JUMPNC   ;19        Has 'A' overflowed?
 9:   3  99   STORE    A;@99      Yes - store it and
11:   2  98   LOAD     A;@98      fetch the hundreds
13:   5   1   ADD      A;1        Add one hundred
15:   3  98   STORE    A;@98      Put the new total back
17:   2  99   LOAD     A;@99      Retrieve the units
19:  16  59   SUB      X;59       Is X-59 zero?
21:   9  26   JUMPNZ   ;26        Jump on if not
23:   3  99   STORE    A;@99      All done, store units
25:   0       HALT                Finished!
26:  15  60   ADD      X;60       Advance X by 60-59
27:  10   6   JUMP     ;6         Round again!
```

**Program 2. A better adder.**

```
 0:  11  90   LOAD     X;90       Point to bottom line
 2:   1   1   LOAD     A;1        1 is data to be moved
 4:  18       STORE    A;@X       Position the '1'
 5:   4       LOAD     A;X        Copy the position into A
 6:   6  99   SUB      A;99       Have we reached the end?
 8:   9  13   JUMPNZ   ;13        Not yet
10:  18       STORE    A;@X       Economically clear the data
11:  10   0   JUMP     ;0         Start from scratch
13:   1   0   LOAD     A;0        Trail a '0' after the '1'
15:  18       STORE    A;@X       Position the '0'
16:  15   1   ADD      X;1        Advance to the next address
18:  10   2   JUMP     ;2         Loop round to put '1' there
```

**Program 3. Moving graphics.**

## TRANSPUT

Our final programming topic this month concerns what is known as 'memory-mapped I/O'. I/O stands for Input/Output, alias communication between the computer and someone or something outside. Followers of the language Algol 68 tried to replace this rather ugly term with the invented word 'transput', but, sadly, it didn't catch on (rather like Algol 68!).

One of the biggest problems for the machine-code programmer is I/O — you can't just say INPUT or PRINT and watch words magically appear on the screen. Most modern computers use memory-mapped I/O, which means that they communicate with the outside world just as they do with their own memory — by storing and retrieving information at certain locations.

The electronics to drive the video display of most micros is quite simple. In effect, a set of memory addresses are connected simultaneously to the computer (which can read and write to the addresses) and to electronics which drives the TV or monitor. The electronics scans through the memory 50 times a second, producing a picture signal for the display. The display is produced by turning a 'dot' of light on and off as it scans across the picture, so that if the dot is on the screen glows and if it is off the screen is dark.

Imagine that the processor stores a selection of numbers in the first half of the display memory, and zeros in the second half. Depending upon the exact electronics used, this will produce a blank screen at the bottom and a jumble of dots or characters at the top. From this you can see that the more memory you allocate to the display, the more dots you will be able to control, and hence the higher the resolution of the display. So far we've assumed a 'Yes/No' value for each dot. If you use still more memory you can add intermediate values to give the effect of

The Easycode simulator takes memory mapping to its logical conclusion — all of the memory is displayed, all of the time. To move a dot across a computer screen you move a value through the screen memory, wiping the old position before each move. We can produce moving graphics in Easycode (slowly), by moving a value through memory in exactly the same way. Program 3 performs this task, moving the value '1' along the bottom of the screen.

STORE Program 3 and play with it. Slow though it is, it demonstrates exactly the technique used in your favourite arcade games. Of course, most shapes are made up of more than one point, but it is easy to see how a group of points could be made to move together. When you get bored with the horizontal movement, change the value at address 1 to 22, and store 11 at address 17. The graphics should now move diagonally. See if you can work out how to make them move up instead of down.

## END OF PART 2

This month we've shown how the Easycode instructions work. If you intend to learn the principles of machine code, it is important that you experiment with the instruction set. Why not write a program to multiply the value in the A register by the value in X? You'll need two locations for the result. As an experiment in indexing, write a program to count all the occurences of a specific value (say, 0 or 1) in Easycode memory. Example solutions will be presented in the next issue of CT.

# The Key to Spectrum Machine Code Success.

Picturesque's MACHINE CODE SYSTEM is used and recommended by professional software writers, yet the excellent documentation and the friendly, easy-to-use programs have been highly recommended for beginners.

**You will only buy one Machine Code System, so buy the best, the one the professionals use.**

## NEW IMPROVED PROGRAMS — MICRODRIVE COMPATIBLE


### ASSEMBLER

Completely self-contained with its own line editor, giving an easy-to-read 40 column tabulated listing. Auto line numbering, line renumbering and auto tabulation make this one of the fastest and easiest Assemblers to use. 5-character label names. Decimal, Hex or ASCII constants. SAVE/LOAD/VERIFY both the listing and Machine Code to cassette/MICRODRIVE/NETWORK. Assemble from cassette or Microdrive into memory for very large programs. Customise to most Centronics printer Interfaces, or RS232 (with Interface 1) for 80 column printout. FAST ASSEMBLY — 1k of code in 7 seconds. Assembler Directives:— ORG, END, DEFB, DEFW, DEFM, EQU, DEFL. (Microdrive and Centronics facilities only operate with 48K machines.)

**£8.50**
INCL. VAT & P&P.


### MONITOR

The ideal tool to help the beginner get started, yet it contains all the commands for the experienced programmer to run and de-bug machine code programs. Inspect and alter memory contents in Hex or ASCII. Breakpoints and full Register display. NOW WITH SINGLE STEPPING through RAM or ROM. Disassemble any part of memory, RAM or ROM. Dec-Hex-Dec number converter. Printer output to ZX printer or via RS232 (with Interface 1) or customise to most Centronics printer Interfaces. General memory management commands include Hex dump, Insert, Delete, Fill and Move. Can reside in memory with the Assembler (48K machines only) to give a complete Machine Code programming system.

**£7.50**
INCL. VAT & P&P.

Programs supplied on cassette with option to Save onto Microdrive (cartridge not supplied.)

Existing owners can obtain the new programs by returning the **cassette only** to Picturesque, along with a cheque/P.O. for **£1.50 per program** (inc. VAT & P&P). New cassettes will be supplied by return of post.

Available from the "SPECTRUM" chain of stores, branches of John Menzies and all good computer shops, or by mail order by sending cheque/PO to:
PICTURESQUE, 6 Corkscrew Hill, West Wickham, Kent, BR4 9BB. Send SAE for details.

PICTURESQUE

---

# Superbase 64

TRANSFORMS THE COMMODORE 64 INTO A FULL-FEATURED AND PROFESSIONAL DATABASE SYSTEM! WITH UP TO 1000 CHARACTERS PER RECORD ON UP TO 4 SCREENS... AND UP TO 128 ITEMS PER RECORD, DEFINABLE AS KEY, TEXT, NUMERIC, CONSTANT, RESULT OR DATE... IN FILES OF UP TO 16M CHARACTERS! SUPERBASE 64 EVEN HAS SPREADSHEET AND CALCULATOR CAPABILITY, CALENDAR FUNCTIONS, EASY INPUT FROM WORDPROCESSOR/DATA FILES, BOTH MENU-DRIVEN AND PROGRAM OPTIONS, SORTING/SEARCHING, FULLY DEFINABLE OUTPUTS... SUPERBASE 64 IS ESSENTIAL IF YOU WANT THE MOST FROM YOUR 64! SUPPLIED ON CBM 1541 DISK WITH EXCELLENT TUTORIAL/REFERENCE MANUAL. OUR PRICE ~~£99.95~~ £88.00!

# VIZAWRITE 64

VIZAWRITE 64 IS A HIGH-PERFORMANCE, LOW-COST WORD PROCESSOR, WITH ON-SCREEN FORMATTING, THAT TAKES FULL ADVANTAGE OF THE 64'S COLOUR, GRAPHICS AND MEMORY FEATURES... AND SUPPORTS VIRTUALLY ANY PRINTER! WITH A COMPREHENSIVE AND EASY-TO-FOLLOW USER MANUAL, VIZAWRITE IS THE ULTIMATE PERSONAL COMPUTER WORD PROCESSOR! AVAILABLE ON CARTRIDGE (~~£89.95~~ £75!), DISK (~~£79.95~~ £68!) OR WITH VIZASPELL (~~£99.95~~ £85!)

# Master 64

MASTER 64 IS A TOTALLY NEW CONCEPT... A COMPLETE PROGRAM DEVELOPMENT PACKAGE, THAT'S AVAILABLE NOW FOR THE CBM 64. MASTER HAS 85 NEW COMMANDS... AND BASIC IV TOO! PLUS PROGRAMMER'S TOOLKIT, MACHINE CODE MONITOR, BUSINESS BASIC, KEYED DISK ACCESS, MULTIPLE SCREENS, USER-DEFINABLE INPUT ZONES, REPORT GENERATOR, 22-PLACE ARITHMETIC, DISK DATA COMPRESSION, DATE CONTROL, SCREEN PLOT, SCREEN DUMP, AND MORE... EVERYTHING YOU NEED TO PROGRAM YOUR 64 TO TOP PROFESSIONAL STANDARDS! (~~£143.75~~ £115!)

MORE SOFTWARE FOR COMMODORE 64:

| | | |
|---|---|---|
| PRACTICALC (DISK/TAPE) ... | ~~£44.50~~ | £41.50 |
| VIZASPELL (DISK) ......... | ~~£59.95~~ | £49.95 |
| DTL COMPILER (DISK) ...... | ~~£114.53~~ | £109.00 |
| DTL COMPILER (TAPE) ...... | ~~£39.95~~ | £36.95 |
| TOOL 64 (CARTRIDGE) ...... | ~~£49.95~~ | £46.95 |

SOFTWARE FOR COMMODORE PET/700:

| | | |
|---|---|---|
| SUPERBASE (700/8096) ..... | ~~£517.50~~ | £425.00 |
| SUPERSCRIPT (700) ........ | ~~£517.50~~ | £425.00 |
| SUPERSCRIPT (8096) ....... | ~~£454.25~~ | £375.00 |
| SUPERSCRIPT (3000-8000) .. | ~~£286.36~~ | £225.00 |
| MASTER (4000/8000/8096) .. | ~~£339.25~~ | £225.00 |

PRICES SHOWN ABOVE INCLUDE 15% VAT, AND ARE CORRECT AT TIME OF GOING TO PRESS. PLEASE ORDER BY POST OR TELEPHONE, USING CHEQUE, ACCESS, BARCLAY CARD OR OFFICIAL ORDER. DESPATCH IS BY SAME-DAY 1ST CLASS POST, AND FREE (EXCEPT FOR OVERSEAS AND CREDIT ORDERS). FULL PRODUCT INFORMATION IS AVAILABLE ON REQUEST. [REF A27]

Calco Software

LAKESIDE HOUSE, KINGSTON HILL, SURREY, KT2 7QT. TEL 01-546-7256

---

# "Run more than ten tasks on a ZX81-FORTH ROM?"

Sure! More than 10 tasks simultaneously and, in some cases, up to 300 times faster! That's what replacing the basic ROM with the new FORTH does for the ZX81 – and more!

The brains behind the breakthrough belong to David Husband, and he's building Skywave Software on the strength of it. Already orders are flooding in and it's easy to see why.

The ZX81-FORTH ROM gives you a totally new system. In addition to multi-tasking and split screen window capability, you can also edit a program while three or four others are executing, schedule tasks to run from 50 times a second to once a year, and with a further modification switch between FORTH and BASIC whenever you like.

The ZX81-FORTH ROM gives you a normal keyboard with a 64 character buffer and repeat, it supports the 16k, 32k, 64k RAM packs, it is fig-FORTH compatible and it supports the ZX printer.

The price, too, is almost unbelievable. As a "fit it yourself Eprom", complete with manual, it's just £25+VAT.
Add £2 p&p UK (£5 Europe, £10 outside Europe) and send your order to the address below.

Skywave SOFTWARE
David Husband
73 Curzon Road, Bournemouth,
BH1 4PW, ENGLAND.
Tel: (0202) 302385.
International +44 202 302385.

# LEARNING FORTH PART 6

Paul Gardner

**Last but not least in our tour around the FORTH language, we get to the real heart of the system with a look at extending the compiler and using machine code routines.**

To round off this series on FORTH programming we'll cover some of the more complicated aspects of FORTH, like how to extend the compiler and implement machine code routines. To begin with, though, we'll see how FORTH is able to handle different 'length' numbers and even numbers in a different base from decimal.

## NUMBER BASES

We are all familiar with numbers expressed in base 10 or decimal, and most of us will have met numbers expressed in binary or base 2. FORTH lets you handle numbers in any base. Try this:

```
16  BASE  C!
```

BASE is a system variable (that only takes up one byte of memory, hence the use of C!) that stores the current number base. If you try doing some ordinary arithmetic now you'll see that the computer is working in hexadecimal, or hex for short. Hexadecimal numbers use the characters A to F to represent the decimal numbers 10 to 15. So in hex

```
9 + 1 = A
F + 1 = 10
```

and so on. FORTH permits you to work in any number base from 2 to 255. For example BASE 36 uses the letters A to Z to represent decimal 10 to 35. So an example of BASE 36 arithmetic would be

```
ALF  BILL  +  .  M40  ok
```

Hexadecimal is probably the most useful alternative to decimal as any eight bit number can be expressed as two letters in the range 00 to FF. We'll use hexadecimal later when we do some machine code programming.

## NUMBER THEORY

All the numbers we have used on the stack so far have been 'single length integers', and each number has occupied two bytes of memory. Within one byte (eight bits) we can store any number in the range 00000000 binary to 11111111 binary, or 0 to 255 decimal. Within two bytes we can hold numbers on the range 0 to 65535 decimal, but we have already seen that we need to store negative numbers as well. For this reason we store numbers in the computer using the following rule: A negative number is stored in the computer with 65536 added to it.

Suppose that our signed integers range between −32768 and +32767, as I've said before. Zero and the positive numbers 1 to 32767 are stored just as they are. The negative numbers −32768 to −1 are stored as the numbers 32768 to 65535, so they start where the postive numbers leave off and carry on up to the largest possible number that can be stored in two bytes. This method of storing negative numbers is called the two's complement form.

We can see from this that two bytes stored in the computer can be interpreted in two different ways, either as a signed number in the range −32768 to +32767 or an unsigned number 0 to 65535. Which interpretation we use will depend on the circumstances.

For example, . (dot) prints a number from the stack as a signed integer but another word U. (unsigned-dot) prints out the number as an unsigned integer. This means different words can interpret the same number in different ways. Any memory handling words like @, !, C@, C! assume that the address on the stack is an unsigned number, as these are no negative memory addresses.

## DOUBLE LENGTH NUMBERS

If we could get the computer to store numbers using 32 bits of memory (four bytes) then we could store signed integers in the range −2,147,483,648 to 2, 147,483,647 (decimal) or unsigned integers in the range 0 to 4,294,967,295 (decimal).

FORTH has a simple and elegant method of recognising double length numbers as they are entered; if a decimal point appears anywhere in a number, then the number will be interpreted as a double length number and either pushed onto the stack or compiled if it is within a colon definition. Typing, for example, 1000000. will put the double length number one million onto the stack. This can be printed out again using the word D. thus:

```
D.  1000000  ok
```

The decimal point used when inputting the number is only there to show that we want this number stored as a double length number. It is not printed out by D. nor does it have any effect on the scaling of the number; 10000.00 would still be regarded as one million by the system, although the position of the decimal point is kept in a variable DPL. A double length number takes up four bytes of the stack with the upper 16-bit half of the number uppermost on the stack.

Most FORTH implementations have only two double length arithmetic operations, D+ (D-Add) and DNEGATE, with one double length comparison D< (D-less-than). Note: Abersoft uses the word DMINUS instead of DNEGATE. With these few commands it is possible to devise more arithmetic operators, for example:

```
:  D-  DNEGATE  D+  ;
1000000.  1.  D-  999999  ok
```

In most cases single length arithmetic is sufficient, but we need to use double length numbers for our next section.

## FORMATTED OUTPUT

In many real applications the type of numerical output produced

by the printing operation .(dot) would not be adequate. For intelligible computer output numbers often need to be printed in meaningful formats: £14.36 for a price or 13/10/84 for a date. FORTH provides a set of operations for building 'specialised' number printing formats. Here are the formatting words:

| | |
|---|---|
| <# | Start a new formatted number string. |
| # | Insert the next digit of the number being printed into the formatted number string. |
| #S | Insert all remaining significant digits of the number into the string. |
| HOLD | Insert the character on the stack into the string. |
| SIGN | Insert a minus sign into the string if appropriate. |
| #> | Terminate the string and leave an address and count on the stack for TYPE to use. |

An example here should make things clearer. This is a price printing operation:

```
:  .£  <#  #  #  46  HOLD  #S  96  HOLD  #>
TYPE  SPACE  ;
```

For example,

```
1543.  .£
```

would print

```
£15.43  ok
```

The sequence of operations within .£ is as follows:

| | |
|---|---|
| <# | Initialise the special buffer for formatted output (this is usually the pad downwards!). |
| # | Converts the last digit (3) into an ASCII code and puts this in the buffer. |
| # | Converts the next digit (4) and puts this in the buffer. |
| 46 HOLD | Puts the ASCII code for . (46) into the buffer. |
| #S | Converts all remaining digits of the number (5 then 1) putting them in the buffer. |
| 96 HOLD | Insert character code for £ (96). |
| #> | Terminates the string and leaves an address and count on the stack for TYPE. |
| TYPE SPACE | Prints out the number and one trailing space. |

Two important points to note here are that the string is built backwards starting with the least significant digit, and that the formatting operation is designed to operate on a double length number. You can convert a single length number to a double length by using the word below:

```
:  S->D  DUP  0<  IF  -1  ELSE  0  THEN  ;
```

(This is already included in some systems.)

One further point is that the double length number must be unsigned for conversion. If we wish to print negative numbers as well as positive ones, then before the initial >#  the number must be converted to a positive one, and the fact recorded for SIGN to use.

Here's an example that incorporates all of this.

```
:  .£  SWAP  OVER  DABS  <#  #  #  46  HOLD  #S  96
HOLD  SIGN  #>  TYPE  SPACE  ;
```

If DABS is not defined on your system, here is a suitable colon definition:

```
: DABS  DUP  0<  IF  DNEGATE  THEN  ;
```

Here's another example that can be used for printing dates.

```
:  .DATE  <#  2  0
DO  #  #  47  HOLD  LOOP
#  #  #>  TYPE  SPACE  ;
```

For example,

```
140184.  .DATE
```

would print

```
14/01/84  ok
```

Many systems come complete with a few number formatting commands provided:

```
D.R   (d,n --)
```

Print a double length number d right justified in a field width of n.

```
.R   (nl,n --)
```

Print a single length number nl right justified in a field width of n.

As a final example we'll see how a set of 'times tables' can be printed for any given number base in a format that is easily read.

```
DECIMAL  ( to make sure you're in the right base)
: TABLES
BASE  C@  1+  1  DO
CR  BASE  C@  1+  1  DO
I  J  *  5  .R
LOOP
LOOP  CR
;
```

So, entering

```
DECIMAL
2  BASE  C!
TABLE
```

will produce

```
 1    10
10   100
ok
```

For example:

```
DECIMAL
6  BASE  C!
TABLES

 1    2    3    4    5   10
 2    4   10   12   14   20
 3   10   13   20   23   30
 4   12   20   24   32   40
 5   14   23   32   41   50
10   20   30   40   50  100
ok
```

## MACHINE CODE

Although FORTH is clearly a very fast language, there are occasions when you may need to write a time-critical routine or when you might want **total** control over the running of the microprocessor. The following examples can only be implemented on Z80-based micros and are written specifically for Abersoft's FIG-Forth for the 48K Spectrum.

If you remember, or look back, to the point in the series where I described the structure of a dictionary entry, you will know that in

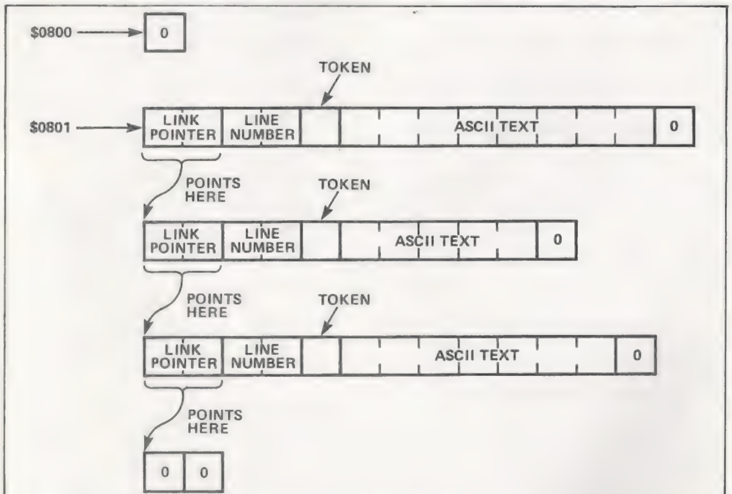



**Fig. 1 The 'linked list' method of storing program lines.**

the header of any word there is two byte field called the code field. This field holds an address for a routine which is executed whenever the word is used. All colon definitions hold the same value in this code field and all variables have a code field that points to the run-time routine for variables, and so forth.

What this is leading to is that the code field pointer actually points to a machine code routine somewhere in the memory and **not** to another FORTH word. This means that if we can set up a word with a machine code routine in its parameter field, we can get the code field pointer to point to the beginning of this routine and then **voila** — we have a word which will execute machine code!

An example is the easiest way to explain. In Abersoft FORTH, CREATE name will set up a dictionary entry, called ' name', with a field pointer that points to name's parameter field. If we now enclose a machine code routine into name's parameter field then executing 'name' will call the machine code routine.

For example, the Z80 machine code for the HALT instruction is 118 decimal, 76 hex. So,

```
CREATE  HALT  118  C,
```

will set up the dictionary header and enclose the code for the HALT instruction into the parameter field. However, we need to know what to do at the end of the machine code routine, so some sort of 'return' routine is called for. Here's a complete definition of HALT for Abersoft FORTH.

```
CREATE  HALT  118  C,  195  C,  108  C,  94  C,
SMUDGE
```

The last three machine code instructions are a call to a routine that will return you to the correct point in the FORTH program that contained HALT, and SMUDGE is an Abersoft word that enables/disables a new dictionary entry from being found during a dictionary search.

Unfortunately, machine code handling is something that is very implementation-dependent so you will have to check your own manual for guidance: most books don't help.

Once a word like HALT has been defined it can be used like any other FORTH word: for example,

```
 :  WAIT-ONE-SEC  50  0  DO  HALT  LOOP ;
```

The HALT instruction suspends execution of any program until the microprocessor receives an interrupt. This is every 1/50th of a second on the Spectrum, so WAIT-ONE-SEC effectively introduces an accurate one second delay in any program that contains it.

As machine code routines are very implementation-dependent I will not dwell much longer on the subject. But, again for Abersoft users, Listings 1-3 provide a very welcome (well, I think so!) routine. If you enter all the words shown in the listing, then entering the commands

```
UTILITY  NEWROUTINE
```

will 'fix' the Spectrum so that every 1/50th of a second the keyboard will be scanned and if you are pressing all three keys, M, SYMBOL SHIFT, and SPACE at the same time then the current FORTH program will be abandoned and the machine will return to BASIC. Typing GOTO 3 will perform a 'warm start', ie return you to FORTH without losing any of the words you've previously defined. This will get you out of 'stuck' programs, like

```
 :  FOREVER
 CR  ."  I  go  "
 BEGIN
 ."  on  and  "
 0  UNTIL  ;
```

Note for FORTH purists; I know that you ought to put an expression like

```
?TERMINAL  IF  QUIT  THEN
```

in a word like FOREVER but you never do, do you?

Note for machine code programmers; clearly this is a way of handling interrupts. If you want to use your own routine then you can define it by following the procedure shown in the example in Listing 4.

```
ROUTINE  name
```

Now enclose your routine using C, and finish your routine with 201 C, after which your new routine can be implemented by

```
name  NEWROUTINE
```

Listing 4 gives a (not very good) way of returning the keyboard bleep to Abersoft FORTH users. If anybody knows a better way, please let me know!

## COLON COMPILATION

We've already met new 'defining words' that enable us to create new data structures like arrays and tables. It is also possible to create new 'compiling words' that allow us to create new program structures. The most obvious examples of compiling words are control-structure words such as IF, THEN, DO, LOOP and so on. It is possible to create your own control-structure words in one of two ways.

Before we continue, we must first review what we know of the colon compiler. In the third article of this series I explained that, when compiling a word, the colon compiler encloses the code field address of a word into the dictionary. There are some words, however, that are not compiled but are executed immediately; these are called 'immediate' words. You can make the most recently defined word in the dictionary immediate by simply typing

```
IMMEDIATE
```

For example,

```
:  HELLO  ."  Hello there !  "  ;  IMMEDIATE
```

Now the word HELLO will be executed whenever it is encountered even if you are in the middle of a new definition. For example:

```
:  TEST  ."  This is a test  "  HELLO  ;  HELLO
THERE !  ok
```

Even during compilation, HELLO is executed immediately. Now whenever the word TEST is executed:

```
TEST  This is a test ok
```

we find that HELLO has not been added as part of the dictionary definition of TEST.

What this is all leading to is to explain that the colon compiler is 'only' another FORTH word, and that some FORTH words are immediate and some aren't. Most of the structure-control words are immediate words. As an example, here's an immediate word we've already met.

```
:  BEGIN  HERE  ;  IMMEDIATE
```

BEGIN simply saves the address of HERE on the stack at compile time. Why? Because another word such as UNTIL or REPEAT needs to know what address to go back to in the event that it must repeat a section of a word. This is the address left on the stack by HERE. Here's an example of a word using BEGIN.

```
:  KEY  BEGIN  INKEY  255  <  UNTIL  ;
```

Figure 1 gives a diagrammatic representation of the dictionary entry of KEY. There are two new points in this diagram we've not met before. The first is that the number 255 is preceded in the dictionary by a pointer to a routine I've called (LITERAL). This word at run time (ie when KEY is executed) copies the number 255 from the dictionary onto the stack. (LITERAL) also arranges that the word after 255 is executed next.

The second point is that a word (UNTIL) is enclosed in the dictionary. This word, at run-time, checks the flag on the stack and if it is false uses the address stored after (UNTIL) to arrange a jump backwards in the execution of KEY. Otherwise if the flag is true it skips over the next entry (the address position) and carries on from there.

The word (LITERAL) is compiled by the colon compiler and does not concern us. The word (UNTIL) though, is compiled by an IMMEDIATE word UNTIL. Here's a definition of UNTIL:

```
:  UNTIL  COMPILE  (UNTIL)  ,  ;  IMMEDIATE
```

COMPILE (UNTIL) compiles the address of (UNTIL) into the dictionary and, encloses an address (the one left by HERE ) into the dictionary.

Well, that's how it works. So here's another structure-control word of our own.

```
:  (JUMPBACK)  R>  @  >R  ;
:  JUMPBACK  COMPILE  (JUMPBACK)  ,  ;  IMMEDIATE
```

This not very useful construction allows you to set up an infinite loop, for example:

```
:  ECHOES  TOHERE  KEY  EMIT
?TERMINAL  IF  QUIT  THEN  JUMPBACK  ;
```

The word JUMPBACK, which is immediate, compiles (JUMPBACK) into the definition of ECHOES and then encloses the address left on the stck by TOHERE as the following entry.

The way (JUMPBACK) works is to take the return address off the return stack. This address points to the next entry in the definition of ECHOES. The contents of this address are fetched and put back onto the return stack. This effectively means that the next word in ECHOES to be executed will be KEY.

If you want to use my cookbook analogy (article three in this series) — if every time you left your recipe to look up a word you didn't understand, someone came along and changed the page number you had written down, then you'd 'return' to a different page to the one you'd left.

Two useful words for when you're defining your own compiling words:

```
COMPILE  ( -- )
```

When a word containing COMPILE executes, the code field address of the word following COMPILE is (compiled) into the dictionary.

```
[COMPILE]  ( -- )
```

'bracket-compile'. When used in the form [COMPILE] name the word 'name' is compiled even if it is an immediate word. For example:

```
:  HELLO  ."  Hello there  "  ;  IMMEDIATE
:  TEST  [COMPILE]  HELLO  ;
```

Now TEST will produce

```
Hello there ok
```

[COMPILE] is most useful when you need to compile a control-structure word like BEGIN and so on. This allows us to invent new control-structure words by adapting some of the other ones.

For example, if you have to implement a multiple decision structure like the example below, you have to use a series of nested IF . . THEN statements, but these can be replaced by one neater structure, the CASE structure, thus:

```
:  PRINTDIRECTION  ( n  --)
   ( take a number off the stack and print the
     direction it corresponds to, if any)
DUP  8  =  IF  ."  NORTH "
ELSE
DUP  6  =  IF  ."  EAST "
ELSE
DUP  2  =  IF  ."  SOUTH "
ELSE
DUP  4  =  IF  ."  WEST "
THEN
THEN
THEN
THEN
DROP  ;
```

This is equivalent to the CASE structure shown below:

```
:  PRINTDIRECTION  ( n --)
CASE
8  OF  ."  NORTH "  ENDOF
6  OF  ."  EAST "  ENDOF
2  OF  ."  SOUTH "  ENDOF
4  OF  ."  WEST "  ENDOF
ENDCASE
;
```

If we work out that every OF is equivalent to the expression

```
OVER  =  IF
```

(consider the stack effects) and every ENDOF is equivalent to

```
ELSE
```

then ENDCASE is equivalent to

```
THEN
```

repeated for the number of OFs in the construct, and a final DROP. Once you've worked this out you can define words such as CASE, OF and so on in terms of the pre-defined words IF, ELSE etc.

Listing 5 gives the complete definitions of all these words, plus two more that are quite handy:

```
WITHIN  (n1,n2,n3  --  flag)
```

Takes three numbers off the stack and leaves a true flag (1) if n1 < n3 and n3 < n2, otherwise leaves a false flag (0).

```
INRANGE
```

An immediate word that uses INRANGE to control part of a CASE construct. For example:

```
:  TEST  ( n  --)
CASE
1  OF  ."  ONE "  ENDOF
2  OF  ."  TWO "  ENDOF
3  20  INRANGE  ."  BETWEEN 3 AND 20 "  ENDRANGE
ENDCASE
```

So to define our own compiling words we can either define them from scratch, like JUMPBACK, or we can define them in terms of other compiler words like IF, ELSE and so on. The final listings give some sort of indication of what sort of structures you can create yourselves.

## CONCLUSION

Well, having got this far in our series on FORTH I think it's time to stop. I like to think I've covered most of the useful and interesting aspects of the language. I know how much I **haven't** covered but that's up to you to discover.

As a final point, FORTH is a unique language, fast, versatile and portable; but without a good supply of software it could become just another interesting oddity. So get writing!

For further reading:

**Starting FORTH** by Leo Brodie, FORTH Inc, 1981. Relatively expensive but enjoyable and thorough.
**The Complete FORTH** by Alan Winfield, Sigma Technical Press, 1983. A good, quite recent book which has plenty of worked examples and a useful pull-out handy reference card.

```
( MACHINE CODE AND INTERRUPT WORDS - GLOSSARY )
( 48K SPECTRUM  ABERSOFT fig-FORTH 1.1A)

ENDCODE finishes all simple m/c words like INTSOFF with a jump
        instruction & allows new word to be found in dictionary.
INTSOFF turns off interrupt routine.
INTSON  turns on interrupt routine.
ROUTINE defines the run-time facilities for your interrupt
        routine. You can use AF,BC,DE,HL & IX registers freely.
        Pressing SYM SH,SHIFT & M at the same time will break
        into forth program. Warm start available by using GOTO 3
UTILITY simple example of use of ROUTINE
KEY-BLEEP provides key-bleep. Change length and pitch by
          altering lines 7 & 8 of listing.
```

```
( LISTING 1 - MACHINE CODE WORDS )
( 48K SPECTRUM  ABERSOFT fig-FORTH 1.1A)
: ENDCODE ( ENDS EACH M/C DEFINITION WITH )
          ( THE SAME EXIT ROUTINE )
  195 C, 108 C,  94 C,    ( JP 24172)
  SMUDGE                  ( ALLOWS SUCCESSFUL )
                          ( DICTIONARY SEARCH ) ;
CREATE INTSOFF ( CREATES HEADER FOR WORD)
   62 C,  62 C,           ( LD A,62)
  237 C,  86 C,           ( IM 1)
  237 C,  71 C,           ( LD I,A)
  ENDCODE
CREATE INTSON
   62 C,   9 C,           ( LD A,9)
  237 C,  71 C,           ( LD I,A)
  237 C,  94 C, ENDCODE ( IM 2)
```

```
( LISTING 2 - DEFINING WORD TO SET UP INTERRUPT ROUTINES  )
( 48K SPECTRUM  ABERSOFT fig-FORTH 1.1A)
: ROUTINE <BUILDS ( SETS UP HEADER FOR WORD )
                  ( THEN ENCLOSES 'BREAK' ROUTINE)
229 C,                    ( PUSH HL)
 33 C,  59 C,  92 C,      ( LD HL,23611)
203 C, 174 C,             ( RES 5,[23611])
225 C,                    ( POP HL)
255 C,                    ( RST 56)
243 C,                    ( DI)
197 C,                    ( PUSH BC)
213 C,                    ( PUSH DE)
229 C,                    ( PUSH HL)
245 C,                    ( PUSH AF)
221 C, 229 C,             ( PUSH IX)
-->
```

```
( LISTING 2 CONT.- DEFINING WORD TO SET UP INTERRUPT ROUTINES)
( 48K SPECTRUM  ABERSOFT fig-FORTH 1.1A)
  1 C, 254 C, 127 C,      ( LD BC,32766)
237 C, 120 C,             ( IN A,[C])
203 C, 247 C,             ( SET 6,A)
254 C, 248 C,             ( CP 248 [SYM SH, SPACE & M])
 40 C,  11 C,             ( JR Z,+11)
205 C, HERE 32 + ,        ( CALL NN[CALL START OF YOUR OWN ROUTINE])
221 C, 225 C,             ( POP IX)
241 C,                    ( POP AF)
225 C,                    ( POP HL)
209 C,                    ( POP DE)
193 C,                    ( POP BC)
251 C,                    ( EI)
201 C,                    ( RET)
-->
```

```
( LISTING 2 CONT.- DEFINING WORD TO SET UP INTERRUPT ROUTINES)
( 48K SPECTRUM  ABERSOFT fig-FORTH 1.1A)
 62 C,  56 C,             ( LD A,56)
 50 C, 141 C,  92 C,      ( LD [23693],A)
205 C, 107 C,  13 C,      ( CALL 3435 [CLS])
  1 C, 254 C, 127 C,      ( LD BC,32766)
237 C, 120 C,             ( IN A,[C])
203 C, 247 C,             ( SET 6,A)
254 C, 248 C,             ( CP 248)
 40 C, 248 C,             ( JR Z,-8)
251 C,                    ( EI)
207 C, 20 C,              ( RST 8 [ERROR BREAK])
DOES> ( AT RUN TIME LEAVES PFA)
      ( OF ROUTINE ON THE STACK)
;
```

```
( LISTING 3 - MORE INTERRUPT WORDS )
( 48K SPECTRUM  ABERSOFT fig-FORTH 1.1A)
ROUTINE UTILITY 201 C,

: NEWROUTINE ( N-)
  INTSOFF          ( TURN OFF OLD ROUTINE)
                   ( NOW PRODUCE CALL INSTRUCTION FOR NEW ROUTINE)
  205 65129 C!  ( CALL)
      65130  !  ( ADDRESS FROM STACK)
  201 65132 C!  ( RET)
;

( EXAMPLE USE OF INTERRUPT FACILITY )

( UTILITY NEWROUTINE INTSON )
```

```
 0 ( LISTING 4 - )
 1 ( THE LONG LOST KEYBOARD BLEEP )
 2 ( 48K SPECTRUM  ABERSOFT fig-FORTH 1.1A)
 3 ( ROUTINE TO RETURN THE KEY-BLEEP FACILITY )
 4 ROUTINE KEY-BLEEP
 5  58 C,  59 C,  92 C,   ( LD A,[23611])
 6 203 C, 111 C,          ( BIT 5,A)
 7 200 C,                 ( RET Z)
 8  17 C,  75 C,   0 C,   ( LD DE,75)
 9  33 C, 150 C,   0 C,   ( LD HL,150)
10 205 C, 181 C,   3 C,   ( CALL 949[BEEPER])
11 243 C,                 ( DI)
12 201 C,                 ( RET)
13
14 ( TO USE THIS ROUTINE TYPE )
15 ( KEY-BLEEP NEWROUTINE INTSON )
```

```
( LISTING 5 - CASE ROUTINE USING DEFINED WORDS )
( 48K SPECTRUM  fig-FORTH 1.1A )

0 VARIABLE NUMCLAUSES ( HOLDS COUNT OF NUMBER OF CLAUSES )
: CASE1 ( CALLED CASE1 TO DIFFERENTIATE FROM PRESENT VERSION)
  0 NUMCLAUSES ! ; IMMEDIATE
: OF1 COMPILE OVER COMPILE = [COMPILE] IF
      1 NUMCLAUSES +! ; IMMEDIATE
: ENDOF1 [COMPILE] ELSE ; IMMEDIATE
: ENDCASE1 NUMCLAUSES @ 0 DO
         [COMPILE] THEN LOOP COMPILE DROP ; IMMEDIATE
: WITHIN ( N1,N2,N3 - FLAG 1 IF N1<=N3<=N2)
       DUP ROT > NOT ROT ROT > NOT AND ;
: INRANGE COMPILE >R COMPILE OVER COMPILE R> COMPILE SWAP
        COMPILE WITHIN [COMPILE] IF 1 NUMCLAUSES +! ; IMMEDIATE
: ENDIN [COMPILE] ENDOF1 ; IMMEDIATE
```

```
( LISTING 6 - AN EXAMPLE )
( 48K SPECTRUM  fig-FORTH 1.1A )

: TESTCASE ( AN EXAMPLE USING THE NEW CASE CONSTRUCTION )
           ( N -)
   CR CASE1
      1 OF1 ." ONE" ENDOF1
      2 OF1 ." TWO" ENDOF1
  3 10 INRANGE ." BETWEEN 3 & 10" ENDIN
  ENDCASE1 ;
```

```
( LISTING 7 - INTEGER CASE CONSTRUCTION )
( 48K SPECTRUM  fig-FORTH 1.1A )

: (INTCASE) R> @ >R ;

: INTCASE COMPILE (INTCASE) HERE DUP , 1 ; IMMEDIATE

: ($) R> @ >R ;

: $ COMPILE ($) HERE DUP , SWAP 1+ ; IMMEDIATE

: (ESAC) DUP 1 < IF CR ." Error.Invalid integer for INTCASE!"
        SP! QUIT THEN
  I SWAP DUP ROT DUP @ 2 / 2 - ROT < IF
  4 + @ SWAP DROP ELSE DUP @ + SWAP 1 - 2 * - @ THEN
  R> DROP >R ;
```

```
( LISTING 7 CONT. - INTEGER CASE CONSTRUCTION )
( 48K SPECTRUM  fig-FORTH 1.1A )
: ESAC COMPILE (ESAC)
  DUP 1 < IF CR ." Error in compilation of INTCASE" SP! QUIT
 THEN HERE SWAP DUP 2 * , 0 DO SWAP , LOOP
 DUP DUP 2 + @ SWAP DUP @ + @ SWAP 2 + SWAP !
 DUP DUP DUP @ + SWAP 2 + DO HERE I @ ! 2 +LOOP
  DUP DUP @ + 2 + SWAP 2 + DO I @ 2 + I ! 2 +LOOP ; IMMEDIATE

( EXAMPLE USE OF INTCASE )
: TESTINT ( N-)
  INTCASE
  ." FIRST CLAUSE " $
  ." SECOND CLAUSE " $
  ." NUMBER SUPPLIED > 2 "$
  ESAC ;
```

```
( LISTING 8 - GLOSSARY OF INTCASE WORDS)
( 48K SPECTRUM  fig-FORTH 1.1A )

The INTCASE structure described here is similar to the Algol 68
case structure, i.e. integer based.

INTCASE at run time expects an integer on the stack, the clause
corresponding to that integer will be executed, then execution
continues from the point after ESAC.

$   The character $ is used to terminate each 'clause' in the
case structure. If no clause is supplied the word being defined
will not compile.

If at run time the integer on the stack exceeds the number of
clauses then the last clause will be executed.
```

**Listings 1-8. Some machine code words and useful FORTH definitions.**


The problem with buying a home computer, as you may already have discovered, is there's often very little software to go with it. Or all that is available is games, games and more games.

There's no such problem, however, with the Commodore 64. It has a more extensive range of serious software than any other home computer.

It also has an unusually large (in fact elephantine) 64K memory, as well as every peripheral you're ever likely to need.

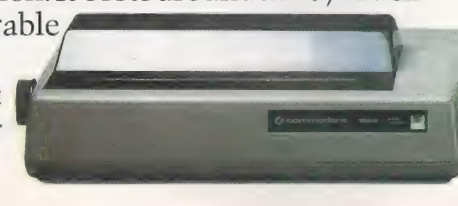
Put simply, this means the computer has the capacity to run more interesting, entertaining and complex programs.

You can teach yourself just about any subject you care to mention, even computer programming.

And for the office there are programs like word processing, financial planning, information storage and stock control.

Finally, when you're mentally exhausted, you can even entertain yourself – yes, with games.

When all's said and done, however, we do have to admit that in one respect the Commodore 64 isn't up with the competition. It costs around £229, much less than any comparable machine.



And that's a fact we hope you'll never ever forget.


# When you have an enormous memory there's no end to the things you can do.

Please send me further information on: the 64 computer ☐ 64 software ☐ disk drive ☐ cassette unit ☐ printers ☐ monitor ☐

Name (Mr. Mrs. Miss) ______________________

Address ______________________

______________________

Send to: The Commodore Information Centre, 675 Ajax Avenue, Slough, Berkshire SL1 4BG. Or telephone (0753) 79292.

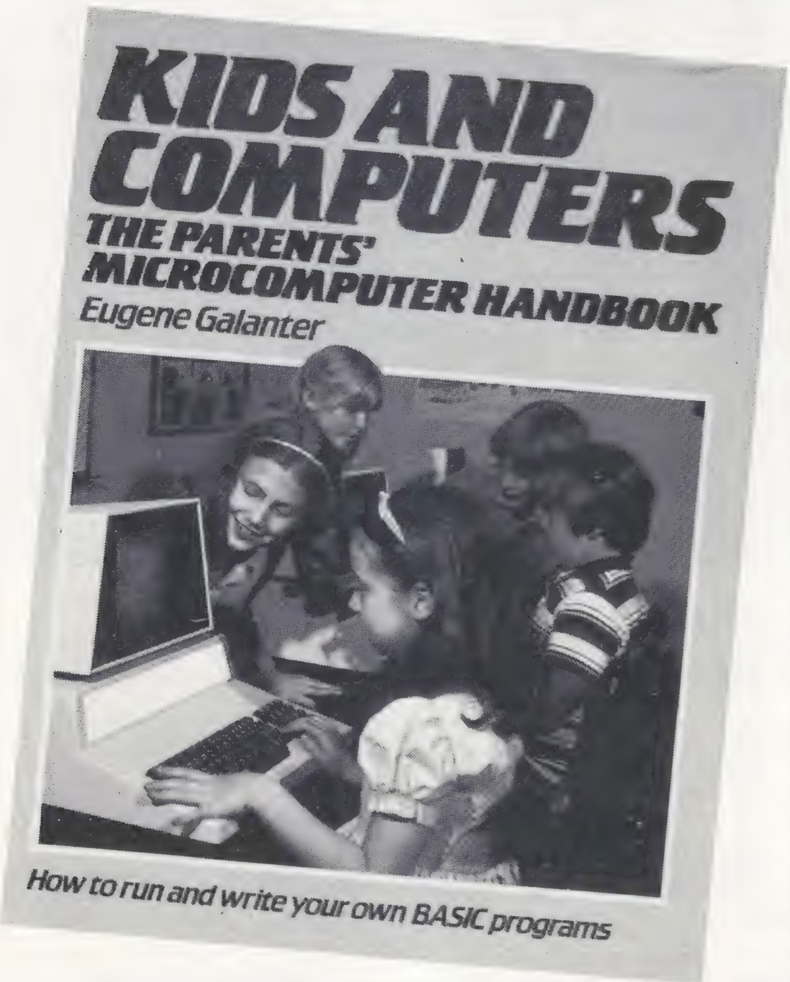
commodore

# BOOK PAGE

Garry Marshall

**Excellence is the order of the month, as we feature a group of books that all get a definite thumbs-up from our reviewer.**

This month's books form a remarkably attractive selection. They are all well-written and address their topics with authority. The first is about what computers are and what they can do, and is aimed at parents, children and teachers. It should provide anyone in these categories having little knowledge of computers with the first steps towards computer literacy. (I have resolved not to use the word 'computerate' again. Having recently met Chris Cunningham who wrote **How to Become Computerate,** it seems proper to treat the word as if it were his.) The second book is a very fine guide to creating games on the computer. It gives the ideas needed to write the programs for a wide range of games and, besides this, it provides an entry into artificial intelligence. Graphics is the subject of the third book, and it is one of the few that uses BASIC as its graphics programming language and so is suitable for microcomputer usage. The last book is called **Computers Today.** With a title so close to that of the magazine, we probably could not avoid reviewing it, but it also happens to be a good textbook on contemporary computing.




**Kids and Computers — The Parents' Microcomputer Handbook** by Eugene Galanter has several interwoven themes throughout its pages. One is to explain at an elementary level what computers are and how they work. The author firmly believes that one way of avoiding any fear of computers that people may have lies in understanding how they do the things that they do. A second theme is to show what computers can do, particularly in education, and a third one concerns the way that computers should be introduced to children. Galanter is particularly well qualified to write on the last theme as he runs a childrens' computer school.

The book was originally published in America, but this is a British edition that has been quite carefully prepared with the help of Brian Reffin Smith. His influence can be detected in several places and, besides this, the spelling is Anglicised throughout the book and British micros are mentioned and evaluated alongside the American ones. The writing style is simple, with few long sentences. This does not imply that the author is writing down to his audience, for he uses a wide vocabulary and is not afraid to use jargon words (with a proper explanation in each case).

Two chapters are almost entirely devoted to explaining how a computer works. They are not at all deep, but their level is appropriate to the book's aims. One chapter gives a block diagram treatment and description, while the other looks at the parts making up each block and introduces hardware items such as the CPU, ROM and RAM in the process.

## BASIC AIDS

What computers can do is partially demonstrated by developing short BASIC programs for arithmetic drills and similar things, but applications such as word processing and information storage and retrieval are described. The author believes that word processing and database programs are valuable educational aids that can be used to advantage by children in and out of school, for example to prepare perfectly-presented essays and written work. However, he also advocates the use of computers, and of programming, as a means of instilling the mental structures and disciplines that were formerly supposed to be acquired by studying subjects such as Latin. (And an infinitely more enjoyable way of acquiring them too, I might say.)

The matter of introducing computers to children is, in some ways, the most important theme of all, particularly as far as our future is concerned. The book asserts that the home computer can contribute more to childrens' intellectual development than any other single item. Programming is intimately involved in this, not that every child should become an expert programmer but that a knowledge of programming allows them to take charge of the computer. Having learnt to program, it is easy to distinguish that gas bills for zero pounds and zero pence and other aberrations are not the computer's fault, and also to appreciate how to use the computer. The book has a very interesting table giving childrens' programming capabilities at ages from five onwards as determined from practical experience. This may be an eye-opener for many people, but it is corroborated by the recently-published **Using Microcomputers In The Primary School** by Peter J. Wayth (Gower). In fact, a number of Galanter's observations and assertions are also confirmed in this book.

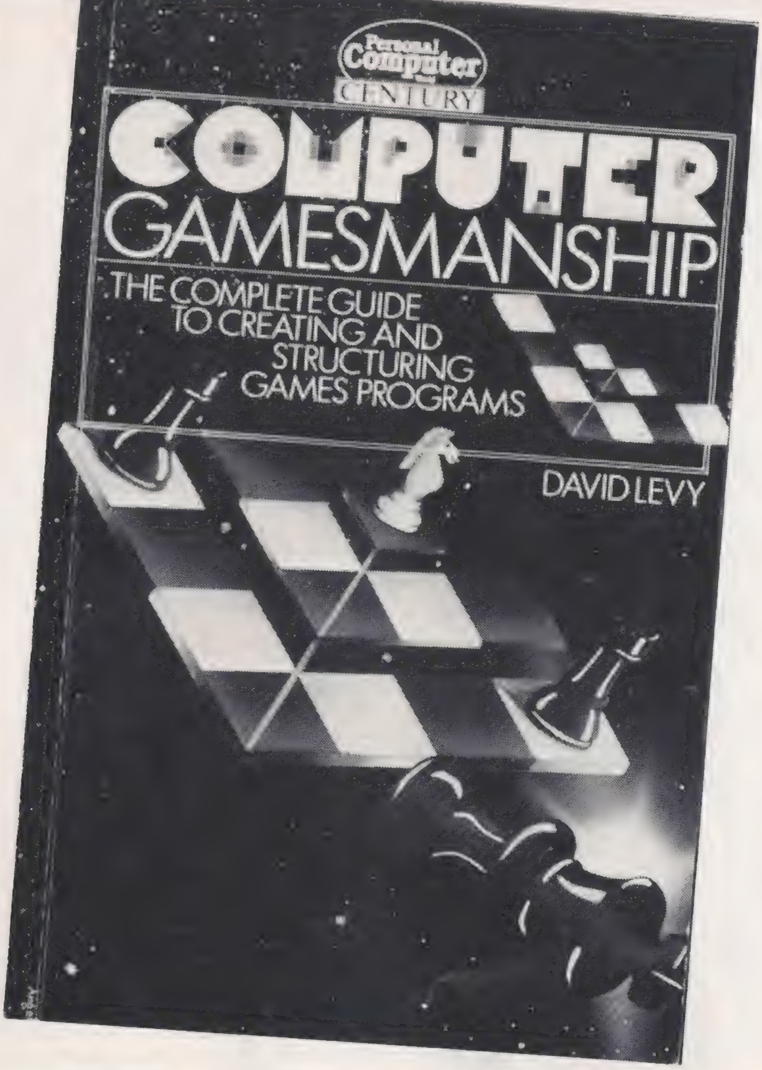
## GIRLS BEAT BOYS

One finding that is particularly interesting in view of the fact that only one or two per cent of the readers of this magazine (and of all the other popular computing magazines, for that matter) are female, is that in Galanter's experience girls are, if anything, rather better than boys at computing. So girls should not feel at any disadvantage when it comes to computing, and there is also no reason why they should not read computer magazines!

I found this a stimulating book, and while it doesn't provide beginners with all the answers about computing, it does give them most of the questions. It gives a very good introduction to computers and computer literacy. My only reservation is that,

when the author ventures beyond introductory matters, his grasp is sometimes suspect in technical matters.

**Computer Gamesmanship** by David Levy is a beautifully written book, based on articles published in computer magazines. Its subtitle is 'the complete guide to creating and structuring games programs'. Let us clear up one or two possible misunderstandings first. It has nothing to do with games such as Space Invaders or Pacman. The games it is concerned with are chess, draughts and similar games of skill. Also, there isn't a single program in the book, although there are several flowcharts. It gives a host of ideas for creating programs, but leaves the reader entirely to his own devices in writing them.




Having got this out of the way, the book itself begins with a chapter on Nim and the 8-puzzle, which are games that are as simple as can be while still displaying all the characteristics of the more complex games. Explanations are given of the vital steps in a game program, which are:

- how to generate moves
- how to assess a position, and
- how to choose a move.

The examples given in this chapter, and throughout the book, are superb. They illustrate exactly what Levy is saying and it is hard to imagine that they could be better chosen.

The second chapter shows how the successive moves and counter-moves that can occur in a game can be represented by a tree. Each path from the root to a leaf of the tree represents a complete game. It is possible to assess how good any position in a game is by giving it a score with an **evaluation function.** The computer is then in a position to decide which move it should make next from any position. One way it can do this is to find the maximum score that its opponent can make in response to each of its possible moves and then to choose the move that leaves the opponent the minimum of all these possibilities. For obvious reasons this is called the minimax method.

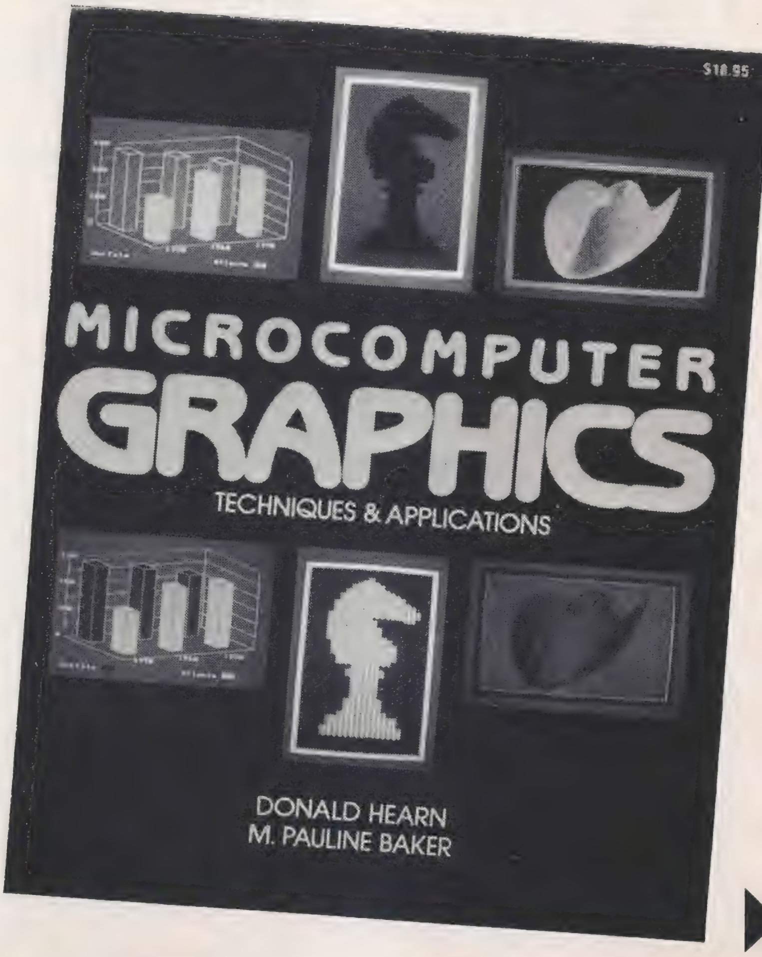
If the entire tree for any game could be stored in a computer, it could always plan the best possible game and would almost invariably win. However, the 'combinatorial explosion' of possible moves ensures that the tree is too big for this. Ways of 'pruning' the tree to a reasonable size must therefore be found. The minimax method is one way, but Levy examines others. He also examines how evaluation functions can be devised using specific information to tailor them to, say, chess or draughts. Having established these principles he, then examines how they can be applied to 11 games ranging from dominoes and draughts to chess and go-moku. All of these chapters are the size of a magazine article except for the 40-page chapter on chess.

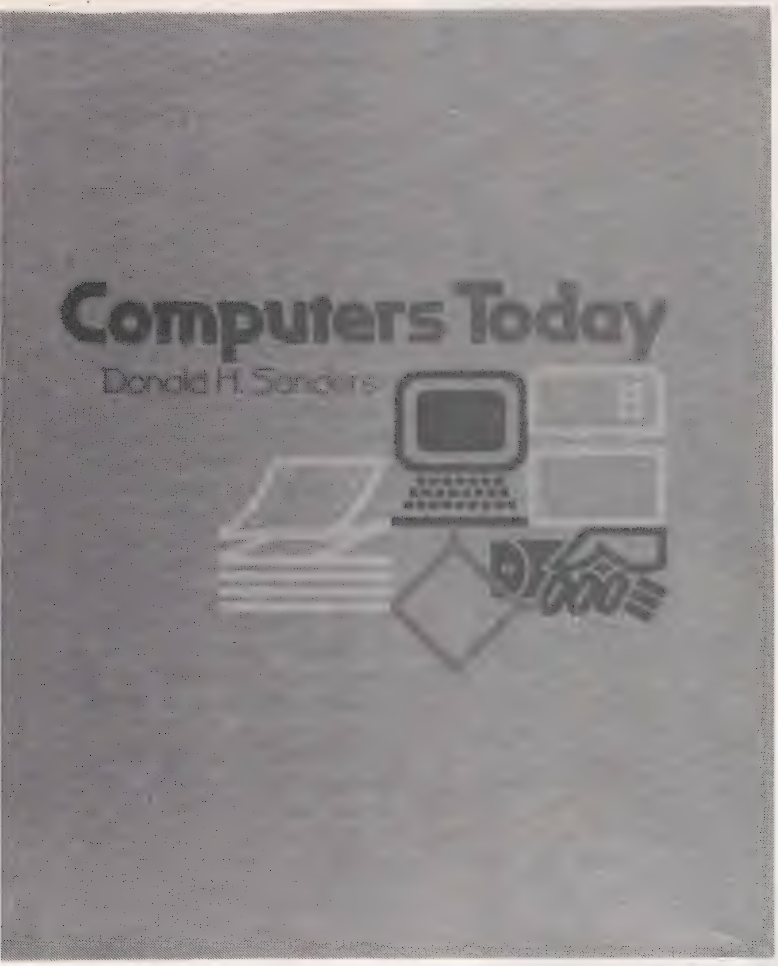
This is an exciting book. It assumes a degree of mathematical sophistication of its readers and leaves them on their own to implement the games programs. It also provides an excellent motivation for studying artificial intelligence. If you have ever read a text on this subject and wondered why it goes on so much about searching trees and generalised problem-solving, this book could give you the insights necessary to tackle such a text again but with renewed enthusiasm.

**Microcomputer Graphics** by Hearn and Baker is an orthodox text on graphics aimed at telling microcomputer owners what they ought to know about graphics. It may not tell them everything they **want** to know, but it certainly covers everything they need to know to have a solid basic knowledge of graphics.

After a general introduction to computer graphics, the graphics facilities of various micros are described. Since this is an American book, only American machines are mentioned. For the graphics programming, an idealised set of graphics commands is added to BASIC. The commands have names such as POSITION, COLOR, DRAWLINE and CIRCLEPLOT which make their functions quite clear. An appendix gives their equivalents in the BASICs of the various micros mentioned earlier.

The treatment of graphics programming starts with image generation using PRINT com-






mands and drawing curves on a high-resolution screen. It then deals with image transformations (rotations and the rest) and animation, before tackling the perspective drawing of three-dimensional objects. It includes a large number of programs and illustrations of the graphics they produce. The final chapter on applications is rather uninspired, relying heavily on graphs and charts for its examples. However, all-in-all this is a good and relevant treatment of graphics for micros.

**Computers Today** by D. H. Saunders is a beautifully-produced book in the tradition of large American text books that is aimed at giving a treatment of the contemporary computer scene. Microcomputers and BASIC have their places in the book, but only within the context of the entire computer scene. The book has five main sections. The first one is introductory and explains quite thoroughly what computers are and what they can do. It covers much the same ground as **Kids And Computers** but in greater depth and more thoroughly. After this, any of the remaining sections on hardware, software, systems, systems and social impact can be read independently. The hardware and software sections are straightforward in their coverages, but the section on systems is primarily about business systems. It deals mainly with file processing and management information systems. The part on social impact is thought-provoking and concludes with some predictions for the future of computing, although I feel that it may take more than the one term suggested by the author to absorb its contents.

This month's books were:
**Kids And Computers** by Eugene Galenter, Kingfisher Books, 189 pages, £5.95.
**Computer Gamesmanship** by David Levy, Century Publishing Co, 272 pages, £7.95.
**Microcomputer Graphics** by Donald Hearn and M. Pauline Baker, Prentice Hall, 302 pages, £16.10.
**Computers Today** by Donald H. Saunders, McGraw-Hill, 669 pages, £19.95.


Programmers: we offer you The World
(which world is up to you!)
Argus Press Software (APS) is seeking to expand it's range of products into all areas of computer software, and we need talented programmers to help us. We already market the highly successful ASP range of adventure and strategy games and will be announcing new ranges in other fields in the near future.
APS is a member of the Argus Press group of Companies and is a sister company to Argus Specialist Publications, Europe's largest computer magazine publishers, (Computing Today, ZX Computing, Games Computing, Personal Computing Today, etc, etc). As such we can offer all the benefits of a large company whilst retaining the flexibility of a smaller, highly motivated unit.
We are looking for both product and people. If you have a piece of software for any of the popular micros that you think is marketable, send it to us at the address below. Our commission structure is second to none and our generation procedure is designed to offer programmers the maximum flexibility.
Alternatively - if you think you have the talent at the keyboard to help us realise our designs - write and tell us what you can do and upon which machine you can do it!
Write now for details to:
Ron Harris
Argus Press Software
No 1 Golden Square
London W1R 3AB
APS

# “ Leaves the Atari, Dra
# and Lynx


Choosing a home computer is a bit like playing a video game. The more you play, the better you get. So you'd expect people who spend their working life choosing computers to be pretty good at the game. Quick to shoot down a fault. Pleased when they find an improvement.

Recently Which Micro? and What Micro? tested the Spectravideo SV 318. Here's what they had to say:

"Every home computer coming on the market tends to be hailed as revolutionary. For once this really has to be true with the Spectravideo SV 318."

"The first cheap, high performance computer..."

"Double precision numbers . . . are ideal for companies with turnovers under £999,999,999,999.99."

---

**Spectravideo SV 318:** Memory – 32K ROM expandable to 96K, 32K RAM expandable to 144K: **Keyboard** – calculator type, 71 keys, 10 function keys, built in joy stick/cursor control: **Graphics** – 16 colours, 256 x 192 high resolution graphics, 32 sprites: **Sound** – 3 channels, 8 octaves per channel: **CP/M* compatibility** – over 3000 existing software packages: **Storage** – cassette drive, 256K disc drive capacity: **Suggested retail price** – £186.

**Spectravideo SV 328:** Memory – 32K ROM expandable to 96K, 80K RAM expandable to 144K: **Keyboard** – full word processor type, 87 keys, 10 function keys, built in cursor control: **Graphics** – 16 colours, 256 x 192 high resolution graphics, 32 sprites: **Sound** – 3 channels, 8 octaves per channel: **CP/M* compatibility** – over 3000 existing software packages: **Storage** – cassette drive, 256K disc drive capacity: **Suggested retail price** – £262.

A full range of peripherals are also available.

** Which Micro Hardware Revue – Spectravideo SV-318

* ® CP/M is a registered trade mark of Digital Research Inc

# POKER

P. J. Kenworthy

In Part 2 of this article we explain in detail the workings of the program we published last month for the BBC Model B.

## LINES 10-80

Initialise the major variables used in the program and dimension those which need to be. The auto-repeat function is disabled (line 30), and the error handler is set to jump to a line (5250) which re-enables the auto-repeat facility (line 10).

## LINES 90-360

Display titles on screen and initiates a musical accompaniment. The arrays are dimensioned and the character set partially redefined. The player is asked if he wishes to play Wild Poker.

## LINES 370-690

Computer runs in Graphics Mode 4. At line 400, the computer checks to see whether 10 hands have elapsed since the player has taken out a loan (assuming one has been taken out). If this is so, the computer runs the routine which decides whether the player can continue or not.

The player's cards are displayed (line 410: see later notes), and the text window set up. The first option to fold is then given to the player. Assuming this is declined the computer then asks how many cards are to be discarded. The number is entered, while at line 580, the routines are used which remove the discarded cards from the screen and choose their replacements. The modified hand is then redisplayed at line 610, together with another option to fold. If this is declined then the program carries on.

## LINES 700-720

The computer evaluates its hand and decides how many cards to replace (see later notes). This result is then displayed on the screen.

## LINES 730-830

Once again the computer evaluates its hand. If the hand has no value at all, then the computer will decide whether to fold or to try and bluff the player out (lines 750 and 760). If the computer decides to carry on or if the hand is of value, then the decision of who should be first is taken. This is normally decided on the contents of V$. If the computer won the last hand then V$ would have "C" as its contents and the computer would be first in the next hand. If the player had won the previous hand, then V$ would be "P" and the player would be first. At the beginning of the game V$ is a null string and the decision as to who should be first is decided by a random number (lines 820 and 830).

## LINES 840-910

If the player has been elected to be first then these lines are used. The starting scores are displayed (line 850) — or up-to-date scores if a game is under way — the player's bet is entered and checked against the "house" rules (lines 870-880). If as a result of the bet, the player's score has dropped below zero, then a loan is offered (line 890). Finally the scores are adjusted and control passes to the computer once more.

## LINES 920-1040

This routine is used by the computer every time it is its turn to bet first. First the weighting routine is run (line 930: see later notes), which assigns a value of the computer's hand, offset against the probability of the player having a better hand. This is then combined with a randomised computation at line 940 to produce a reasonable bet. This is then displayed and the scores adjusted accordingly.

If the player made the first bet, the above routine is skipped. Instead, the computer decides whether it should fold or carry on as a result of the player's bet. If the player's bet is high and its own hand is not of a high standard, then there is a probability of one-third that the computer will fold (line 990). Also, if the computer's score is low and the above two criteria are also satisfied, then it will fold. At line 1030, the decision of whether to see the player's bet is taken. If the computer decides against doing this, it will then automatically go on to raise the bet.

## LINES 1080-1140

If either the player or the computer has elected to see each other's bet, this routine is used. First the player's cards are displayed, then after a text window has been set up (line 1120), the computer's cards are displayed.

## LINES 1150-1450

Between these lines the computer will begin the check to determine who has won. First the player's and computer's hands are evaluated and a weighting attached to them (lines 1150-1170). The variables SCORE (1) and SCORE (2) hold the total value of the player's and computer's hands. This information alone is normally enough to determine who has won a hand. However in certain situations (eg when the player has two pairs and the computer has three of a kind) when SCORE (1) would equal SCORE (2) — both being equal to two in the above example — a further indication of how a hand is made up is needed. This is provided by PS(1) and PS(2). In the above example PS(2) would equal 3, while PS(1) would only equal 2. The lower the value of PS( ) indicates the greater value of hand (see later notes for how this works), so the computer would be declared the winner.

Line 1180 is only used in normal poker. Since runs and straights are allowed in this version, the computer needs special instructions on how to cope with them. These situations are looked for in this line (the reason being that unless this was done, the computer would declare the hand worthless since the evaluation routine only looks out for pairs of cards and so on).

The scores are "weeded out" over the next couple of lines, with a result that by line 1210 only a hand which might contain a run or a flush is left.

The next stage occurs between 1330 and 1440 where the computer determines whether either hand possesses a flush, a straight flush, or a straight (see later notes for how this is achieved). The scores for each hand are changed if necessary. The program then returns to line 1230, where in the next two lines the computer again checks to see if either the player or the computer could be named the winner of the hand. If no conclusion is reached, ie both hands are worthless, then the computer uses the 'Ace-high' routine between lines 1250 and 1320, which merely searches for the highest card possessed by the two players. Should both have the same card as their highest card then it will look for the next highest, and so on until a decision can be reached.

## LINES 1460-1640

The routines between these lines are used to determine the winner if the values of the computer's hand and the player's hand are the same. For example, this would occur if both had three of a kind in their respective hands. The only point to note between these lines is at line 1380, which once again checks PS( ) for both players. If either of these values is over 2, which would make the comparison between SCORE(1) and SCORE(2) possibly ambiguous (see above), then a separate routine is called upon. If this check proves that the normal routine can be carried out, the computer will then check the value of the winning cards from each hand against each other, the more valuable hand being declared the winner.

This decision making is carried out between lines 1510

and 1540, and is done by comparing the value of the first two characters of the strings containing the player's and computer's winning cards. These strings have their contents assigned to them in the evaluation routine (see later notes) and at line 1470. In the unlikely event of both the computer and the player having the same value cards as their winning cards (ie both have two tens in their hands, for example), then the computer will use the ace-high technique once more to work out the winner of the hand (lines 1550-1630).

## LINES 1650-1870

Between these lines, the final decisions are taken as to who has won a hand if both the computer and the player have high scoring hands. Lines 1650 and 1660 decide the difference between one side having two pairs and the other having three of a kind, the winner being announced as a result of this comparison. This leaves two (or perhaps more if wild poker is being played) possible other situations which could arise. The first of these is if both sides have two pairs in their hands, and the other is if one side has four of a kind while the other has two pairs. (The computer is unable to decide between the two at this point in the proceedings).

Between 1800 and 1870 the computer makes a much finer comparison between the two hands which results in the above-mentioned anomalies being sorted out. The winner is finally declared.

## LINES 1880-1930

The winner is announced at these lines, with the contents of V$ being fixed accordingly to determine who will bet first in the subsequent hand.

## LINES 1940-2150

If earlier in the program, the computer made the decision to raise the player's bet, then this routine is used. The computer computes by how much it should raise the player (line 1940), tells the player its decision and offers the player the three options open to him — fold, see or raise the bet still further. The player's choice is entered at line 2010 and the appropriate routines are run, the scores being adjusted

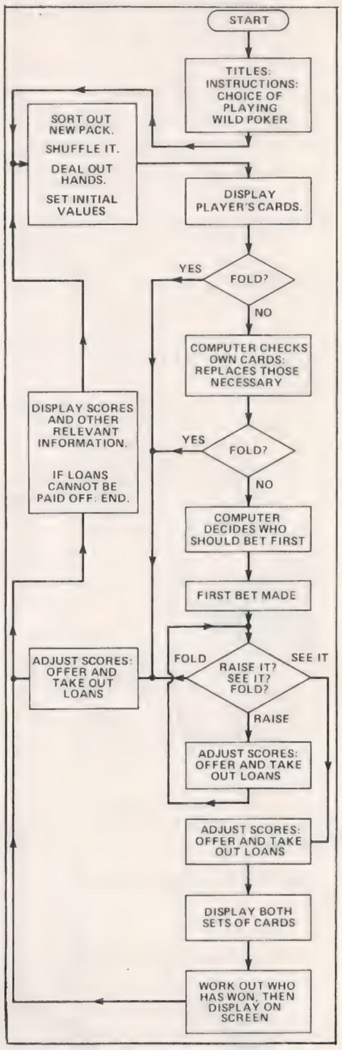



**Fig. 1 A simplified flowchart of the poker simulation.**

accordingly. If the player has elected to raise the computer's last bet, another check is carried out at line 2140 to see whether a loan is necessary. A similar check is made if the player has decided to see the computer.

## LINES 2160-2200

Routine which prints out the player's score, computer's score and the pot while betting is in progress.

## LINES 2210-2280

This is the first of the three routines which are called upon in the event of a possible straight or flush being present in either hand, which would alter the scores of either hand significantly.

All three of these routines use the dummy string S$( ) which is set equal to either the computer's cards or the player's cards depending on the stage which the program has reached.

This routine detects the presence of a full or partial flush (ie the hand wholly or partially consisting of cards of the same suit). This is achieved by looking at the last two letters of each card. The computer checks the last two letters of S$(1) (ie the first card of the hand) against those of all the rest of the cards in the hand, in-crementing a counter (SUIT) if they are the same. The score is incremented by a small or large amount at the end of the routine depending on the final value of this counter.

## LINES 2290-2390

This is the second of the 'value' routines, used for detecting the presence of a whole or partial straight. Use is made of the ascending sort routine which sorts the cards by face value. Like the flush routine, a comparison is made of the value of the first two characters of the first card and then of those following it. The score is similarly incremented according to the value of the counter (Z in this case).

## LINES 2400-2470

The last of the above-mentioned routines, which checks to see if a Royal straight is present (ie Ace, King, Queen, Jack, 10). This is done by adding all the values of the cards together. For a Royal straight, this must be 47. Before the score is adjusted, an additional check is made to ensure that the second highest card in the hand is a King, to reduce significantly the chance of a fluke occurring.

## LINES 2480-2580

Procedure enabling a loan to be taken out by the player if he desires. If this is so, £1000 is added to his score and a counter is set into operation, allowing 10 hands to be played before the loan has to be repaid.

## LINES 2590-2650

Routine used to check if the loan can be paid back now that the 10 hands have elapsed. If this can be done, the loan counter is disabled, the scores re-adjusted, and control passed back to the main program.

## LINES 2660-2740

Computer's equivalent to 2480-2650.

## LINES 2810-3080

Used by the computer to work out how many cards it is going to replace in its hand. This is achieved by first calculating the value of the hand (line 2830), and as a result of this exercise, replacing those cards necessary. This will be explained in greater detail in a moment. If the result of the evaluation routine shows that the hand is, at present, worthless (checked for at lines 2840 and 2850) then the hand is sorted into face value order and a routine performed between lines 3090 and 3160, which replaces either the first two or first three cards, the decision of which being taken by a random number (line 3090).

Assuming that the hand is of value, this part of the program is ignored and the rest of the main routine used instead. To explain how the main routine works, let us assume that the computer's hand consists of three fives (naming a value at random), and two other cards which are at present useless as far as the computer is concerned. The evaluation routine (see later notes), assigns a value PP( ) to each card. In our example, the first five the computer came across would have a value of two assigned to the corresponding variable PP( ), since there are two other cards of similar value in the hand.

Between lines 2860 and 2910, the variables PP(1) to PP(5) are examined. When a value of PP( ) has been found which is greater than zero, the routine is skipped out of (line 2890) and the next part of the process begun. This is between 2920 and 2990, the main purpose of which is to examine how many cards of like value are held in the hand, and to subtract this value from five, which leaves the number of cards which the computer has decided to replace. This value is asigned to the variable EE.

Next, the computer replaces the cards which are not of the same value of the cards upon which the bet is hopefully going to be made. This is checked for at line 3040. An additional check is made in the previous line, namely that if Wild Poker is being played, the computer does not go and replace a wild card by mistake!

The actual changing of the cards is done at line 3050, the workings of which will be explained in a later section.

## LINES 3170-3290

This routine sorts the cards into order by face value. This is achieved by comparing the value of two adjacent cards. If the value of the lower one is higher than the other card, then the two are swopped around (line 3280). If an Ace is found, or a two if wild poker is being played, they are automatically assumed to be more valuable than the next card, consequently being moved up the hand. The complete process is repeated five times to ensure that all the cards have had time to move to their correct positions in the hand.

## LINES 3300-3440

This routine, which is only used in Wild Poker, adjusts the value of the hand if a deuce is detected in it. First the routine searches through the various values of PP( ). If one of these is found to be greater than zero (thus indicating there are two or more cards of the same value) then the deuce is credited to this particular value of PP( ). If all values of PP( ) are zero (no two cards are of the same face value), then the computer sorts the cards into ascending order, and assigns the deuce to the most valuable (lines 3410-3430).

## LINES 3450-3480

Increments the pack variable (the number of the top card of the pack) by one, then replaces a chosen card from the computer's hand by this new card.

## LINES 3490-3840

For this next part of the program structure, it is probably necessary for the explanation to be quite detailed, since it is entirely to do with the on-screen display.

First the basic shapes of the cards are drawn. This is done between lines 3500 and 3530 and is achieved in the following way. Each of the cards is made up of a 5 by 9 grid. In order to draw these on the screen two FOR-NEXT loops are used. The loop at line 3510 draws a rectangular block (using a redefined character 228) five times across the top of the screen. The line feed loop starting at line 3500 is then incremented by one, and the whole process is done again. This carries on, until the five rectangles are each nine lines deep.

The next part of the display routine draws the correct symbol (heart, club, diamond or spade), in the correct places on the screen depending on the value of the card. This is done by making use of a co-ordinate system for each card. The data statements at lines 3820 and 3830 contain the coordinates of where the graphic symbols should go on the card for every non-picture card.

An example should make things clearer. Let us use as an example a card such as the five of diamonds. Between lines 3630 and 3660, the computer checks the last two letters of the string containing the card. In this example, it is DS, so an arbitrary string E$ is made to be the same as the character 225, the code for the diamond shaped graphic-symbol. The value of the card is 5, which means that the computer will pick AA$(5) for the co-ordinates of the card. AA$( ) is defined between lines 3550 and 3570.

AA$(5) will be "2224-538284". This is then gradually broken down into the separate co-ordinates (lines 3680-3710). In this example these will be 2,2: 2,4: 5,3: 8,2: 8,4. The TAB (X,Y) function uses these co-ordinates to print the graphics symbols in the correct position for each card. The diagram in Fig. 2 indicates the co-ordinate system.

The FOR — NEXT loop which commences at line 3600 makes the above process happen five times, thus drawing the player's whole hand.

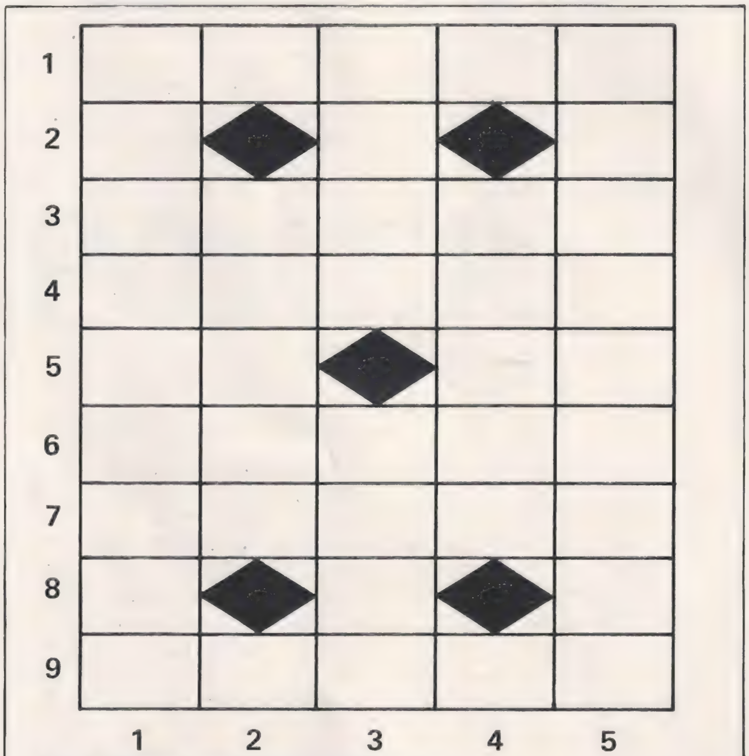



**Fig. 2 Plotting grid for the card display.**

There are a few more points to note about the drawing of the cards. The first of these is at line 3590, which simply reverses the foreground and background colours, thus making the symbols appear as black on a white background.

The second point to note is that the value of a card (with a few exceptions) is printed in the top left-hand and bottom right-hand corner of each card. This is done at line 3740 for values 2-10. If the card is an ace, then "A" is printed instead (line 3730). If the card is a picture card, the symbol of the suit is printed in the two corners.

The final point of importance concerns the picture cards themselves. I decided against a full picture representation of a royal card for the simple reason that my degree of artistic talent is limited, to say the least! Perhaps a reader might like to try to produce a suitable set of user-defined characters.

Instead, the computer prints either KING, QUEEN or JACK in the centre of the relevant card.

## LINES 3850-3900

Adjusts either player's or computer's score, depending on who has just folded.

## LINES 3910-3990

By making use of the TAB (X,Y) function, this routine makes a card disappear from the screen while the player is telling the computer which card he is replacing. Depending on what card the player is replacing, the removal process starts in any one of five positions across the screen (A%).

## LINES 4000-4030

By incrementing the pack variable NN by one, the next card for the player (to replace the one he has chosen to discard) is chosen.

## LINES 4040-4160

This set of instructions is for the general print-out of scores and other relevant information at the end of each hand. If the player has taken a loan out, he/she is informed here of the number of games remaining before it must be repaid. If 10 games have elapsed since the computer, or the player, took out their loans, it is from here that the relevant routines are run (lines 4120-4150).

## LINES 4170-4390

Routine for putting instructions on screen.

## LINES 4420-4690

The main initialisation is achieved between these lines. The statement *FX 15,1 used in line 4430 flushes the contents of the input buffer. Between the lines 4460 and 4530, the cards are mapped out into the data array B$( ). The way this is done is fairly self-explanatory from the listing. The "shuffling" of the cards is achieved at 4540 to 4600. Three numbers between one and 52 are chosen at random (line 4550). The corresponding values in the string array B$( ) are then interchanged with each other, the method of which can be seen in the listing. The string

variable H$ is merely a dummy variable used to prevent one of the cards from being wiped out as the interchanging progresses.

Lines 4610 to 4670 are used to deal out the player's and computer's hands. The pack variable NN is set to one (line 4610), which informs the computer that the first card to be used will be B$(1). The cards are then dealt alternately between P$( ) and C$( ), the pack variable being incremented by one after each card has been dealt (lines 4640 and 4650).

The last part of the initialisation routine at line 4680, subtracts the starting stake (£30) from both the player's and the computer's scores, and sets the pot at £60.

## LINES 4700-4960

These lines probably contain the most important routines in the whole program, enabling the computer to work out the value of its own hand, together with that of the player's when necessary. The mechanics of the routine as a whole are fairly complex and hence rather difficult to explain briefly. Since the working involved in the routines is in standard BASIC there should be no difficulty in its implementation to other systems. However, the end result of the various routines is as follows.

First the variables P(1) to P(5) are produced, which contain the information as to which of the cards are the ones upon which the bet is to be made. The exact value of each variable indicates how many cards there are of the same face value in the hand. For example, if P(1)=2, this would mean that there were two other cards of the same face value as the first in the hand.

The next variable to be produced is B. This indicates the value of the hand as a whole. It is upon this variable that the weighting attached to the amount the computer is going to bet is made. The value of B takes into account the difference in hand values depending on whether Wild Poker is being played or not.

The last product of these routines is the string array H$(N). This string contains the card(s) upon which the bet/evaluation is to be made. The value of N is normally one or two, since the bet is normally made on either one or two sets of cards (ie either on a single pair etc, or on a full house/two pairs).

## LINES 4970-5240

These are used by the computer to produce a weighted evaluation of its hand. It works by the following method. The variables d1, e1, f1 contain the approximate probabilities of getting three of a kind, two pairs and one pair respectively. (The probabilities of getting a hand higher than this were too small to make any difference in the calculations to follow). The variables d,e and f contain the totals of these probabilities. To make this clearer, let us take the probability of obtaining a pair of cards which have the same face values of cards in the pack, there are four cards with the same value. From this it can be seen that there are 26 possible ways of having a pair of cards with the same face value. The variable f is the product of f1 (the probability of a single pair) multiplied by 26. An identical process was used to produce the values d and e.

The way that the computers uses these values shall now be explained. Let us suppose that the computer has in its hand a pair of fives. There are 18 possible pairs which could beat this. On top of this, there exist all the possible hands with two pairs, together with all those containing three of a kind. (NB. Although pure mathematicians are probably shuddering at the suggestions of probabilities in excess of 1, the process does work, and for the purposes of this program the precise rules have been ignored).

For our hypothetical hand containing two fives, the following process is now obeyed. From the variable f, 18 times f1 is subtracted (all the possible pairs which could beat two fives.) To this new total (CC in the program) the variables d and e are added. This value CC, is then subtracted from 3.44 (the sum of d,e and f) to produce the weighting value. These processes take place between lines 5010 and 5180. The ON GOTO statement at line 5020 is used in order to miss out the parts of the routine which are not needed. For example, if the computer had three of a kind in its hand, the weighting processes attached to two pairs and one pair would be ignored.

The variable B, which was earlier obtained from the evaluation routine, is modified by these processes to form a much more accurate picture of the computer's hand, for it to make its bet upon. The computer automatically takes into account whether the game being played is Wild or normal Poker. If Wild Poker is being played, the routines between 5210 and 5230 take into account the presence of a deuce and adjust the weighting variable B accordingly.

## LINES 5250-5270

In the event of an error occurring, or if the Escape key is pressed, control passes to line 5250, which issues an error message and the line in which the error occurred. The auto-repeat facility is re-enabled in line 5260, and the END statement is given, terminating execution of the program and returning control to the user.

## LINES 5280-5300

This procedure produces a sound to indicate that a key has been pressed.

## LINES 5310-5430

This procedure produces the musical accompaniment to the opening title page. The envelope in line 5330 controls the sounds, giving a 'honky-tonk' piano effect to the music.

RESTORE 5440 (line 5340) instructs the computer that any following READ statement is to start reading the data from line 5440 onwards. Line 5380 is designed such that on every sixth note played, the note is played 'staccato' an extra number of times. This is to impart a 'jangle' to the end of each bar.

The tune continues to be played until the 'Y' or 'N' keys are pressed, at which time the procedure is ended.

If the tune finishes, the computer waits for a few seconds. If neither the 'Y' or 'N' key is pressed during this time, the tune is repeated.

## LINES 5440-5480

These lines contain the data for the introductory music.

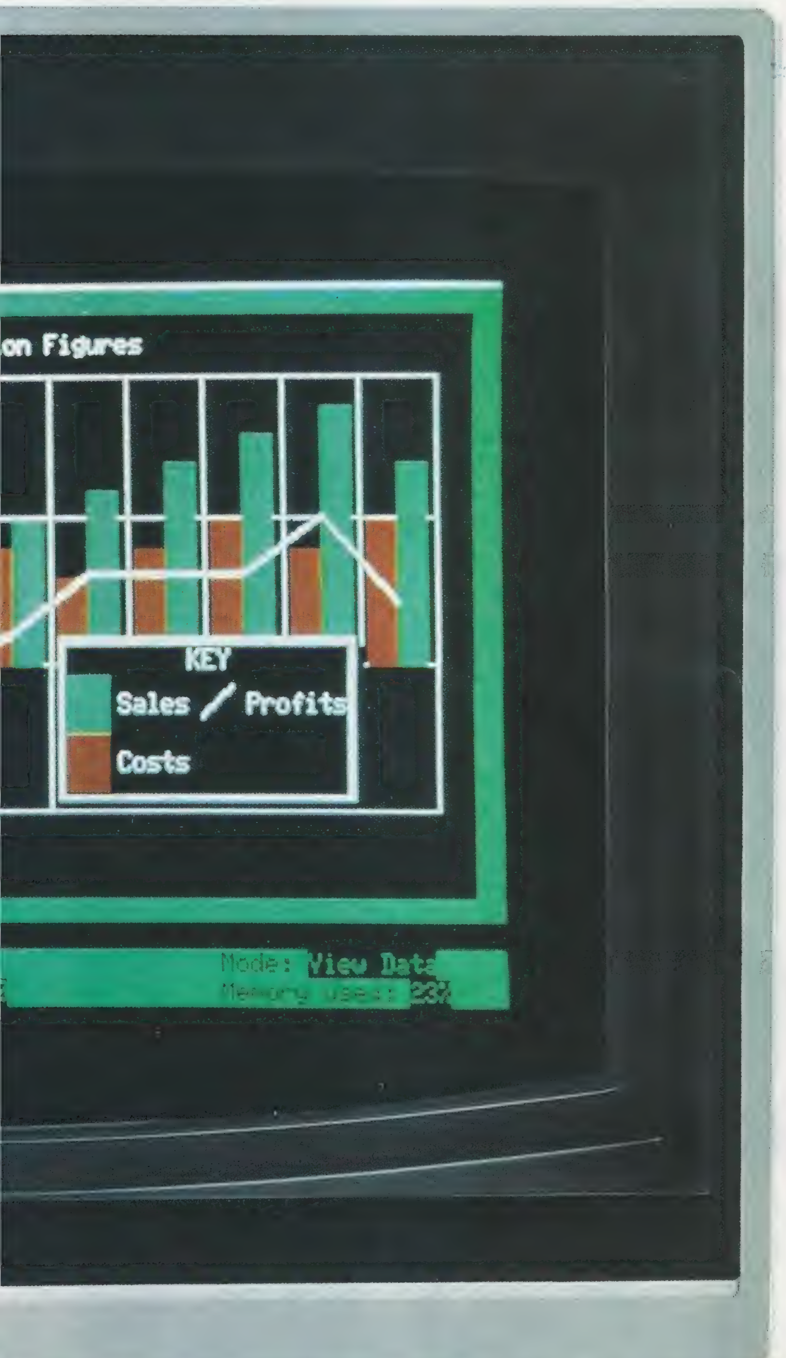
## ADJUSTMENTS

Apart from the improvements to the graphics that I spoke of earlier, the only adjustment the use might wish to make is to the sensitivity of the computer's decision to fold. These adjustments would have to be made in line 990 (if a bet has just been made by the player) and to 750 and 760 for the occasions when the computer is to bet first. In the latter two lines, it would merely be necessary to adjust the figure following the statement RND(4)<. To make the computer more sensitive increase the number there, and vice-versa.

The volume of the music can be adjusted by altering the thirteenth parameter of the envelope statement in line 5330.

# New-Sinclair QL

# There's no comparison chart, b




**The Sinclair QL is a new computer.**

**Not just a new Sinclair computer, but a totally new sort of computer – nothing like it exists anywhere.**

**It's not just a bit better than this, or a bit cheaper than that – it's a computer that's very hard to compare with anything. Just check the features below – and if you don't agree, take up the challenge at the end of the advertisement.**

**If you do agree, there's only one course of action you can take... get yourself a Sinclair QL at the earliest possible moment.**

## The Sinclair QL has 128K RAM. Big deal?

Several micros offer 128K RAM, or more, as standard. The 'What Micro?' table for December 1983 lists over 50 of them – but 40 of the 50 micros listed cost over £2,500!

The Sinclair QL offers you 128K RAM for under £400, and an option to expand to 640K. That's a lot of bytes to the pound!

## The Sinclair QL has a 32-bit processor. Who else?

Under £2,700, nobody. Even the new generation of business computers, such as the IBM PC, are only now beginning to use 16-bit processors.

At prices like this, the Motorola 68000 family – widely regarded as the most powerful microprocessors available – will remain a luxury.

Yet with the Sinclair QL, the 32-bit Motorola 68008 is available for less than £400.

You can also be sure that the QL will not become outdated. 32-bit architecture is future-proof.

**32-bit processor architecture, 128K RAM, and QDOS combine to give the QL the performance of a mini-computer for the price of a micro.**

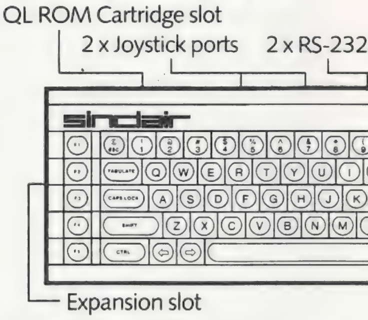
## Exclusive: new QDOS operating system

No competition! QDOS sets a new standard in operating systems for the 68000 family of processors, and may well become the industry standard.

QDOS is a single-user, multi-tasking, time-sliced system using Sinclair's new SuperBASIC as a command language.

One of its most significant features is its very powerful multi-tasking capability – the ability to run several programs individually and simultaneously. It can also display the results simultaneously in different portions of the screen. These are features not normally available on computers costing less than £7,000.

### Eleven input/output ports




## New professional keyboard

The QL keyboard is designed for fast input of data and programs.

It is a full-size QWERTY keyboard, with 65 keys, including a space bar; left-and right-hand shift keys; five function keys; and four separate cursor-control keys – key action is positive and precise

A membrane beneath the keyboard protects the machine from dust (and coffee!), and for users who find an angled keyboard more comfortable, the computer can be raised slightly at the back by small detachable feet.

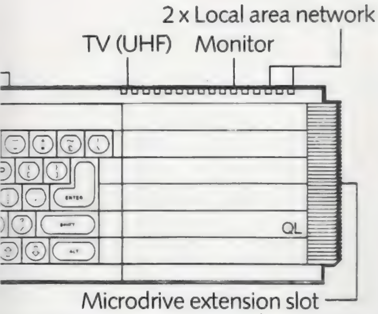
# £399

# ecause there's no comparison!

## Advanced new friendly language – Sinclair SuperBASIC

The new Sinclair SuperBASIC combines the familiarity of BASIC with a number of major developments which allow the QL's full power to be exploited.

Unlike conventional BASIC, its procedure facility allows code to be written in clearly-defined blocks; extendability allows new procedures to be added which will work in exactly the same way as the command procedures built into the ROM; and its constant execution speed means that SuperBASIC does not get slower as programs get larger.




## Two 100K microdrives built in

The Microdrives for the Sinclair QL are identical in principle to the popular and proven ZX Microdrives, but give increased capacity (at least 100K bytes each) and a faster data-transfer rate. Typical access speed is 3.5 seconds, and loading is at up to 15K bytes per second. The Sinclair QL has two built-in Microdrives. If required, a further six units can be connected.

Four blank cartridges are supplied with the machine.


ⓉⓂ Quill, Easel, Archive and Abacus are trade marks of Psion Ltd.

## Included – superb professional software

**The suite of four programs is written by Psion specially for the QL and incorporates many major developments. All programs use full colour, and data is transportable from one to another. (For example, figures can be transferred from spreadsheet to graphics for an instant visual presentation.)**

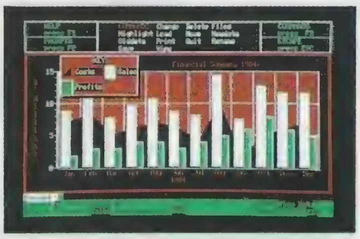
### Word-processing

Certain to set a new standard of excellence, QL Quill uses the power of the QL to show on the screen exactly what you key in, and to print out exactly what you see on the screen.

A beginner can be using QL Quill for word-processing within minutes.

QL Quill brings you all the facilities of a very advanced word-processing package.

### Business graphics



QL Easel is a high-resolution colour program so easy to use you probably won't refer to the manual! It handles anything from lines, shaded curves or histograms to overlapping or stacked bars or pie charts. QL Easel does not require you to format your display before entering data; it handles design and scaling automatically or under your control. Text can be added and altered as simply as data.

### Spreadsheet


QL Abacus makes simultaneous calculations and 'what if' model-construction easier than they've ever been. Sample applications are provided, including budget-planning and cash-flow analysis. QL Abacus allows you to refer to rows, columns and cells by names, not just letters and numbers. Function keys can be assigned to change a variable and carry out a complete 'what if' calculation with a single key-stroke.

### Database management

QL Archive is a very powerful filing system which sets new standards, using a language even simpler than BASIC. It combines ease of use for simple applications – such as card indices – with huge power as a multi-file data processor.

An easy-to-use labelling facility means that you don't have to ask for your file by its full name – a few letters are enough.

## New – the Sinclair QLUB

The QLUB is the QL Users Bureau. Membership is open to all QL owners. For an annual subscription of £35, QLUB members receive one free update to each of the four programs supplied with the QL, and six bi-monthly newsletters. Sinclair has also made exclusive arrangements for QLUB members to obtain software assistance on QL Quill, Abacus, Archive or Easel by writing to Psion.

## The Sinclair QL challenge

If you're seriously considering any other computer, post the coupon for a blow-by-blow comparison. We'll take a *published* comparison chart for the machine you're considering (not one we've created ourselves) and give you the Sinclair QL figures, detail by detail.

## Take action today!

### To order by mail

– complete the coupon and send it to the FREEPOST address below. For credit card holders it may be possible to extend your credit limit. Full details will be sent when we acknowledge your order.

### To order by telephone

– phone Camberley (0276) 685311;
– have your credit card (Access, Barclaycard, Trustcard) number ready. It may be possible to extend your existing credit limit. Please ask our telephone staff for more details. **Please do not use this number for other enquiries.**

### For more information

Phone Camberley (0276) 686100, or use the coupon to get a QL brochure.

Due to demand, delivery may take more than 28 days. Your order will be acknowledged immediately with an expected shipment date. Remember that Sinclair offers a 14-day money-back undertaking.

Send to: Sinclair Research Ltd, Computer Division, FREEPOST, Camberley, Surrey, GU15 3BR.

| Qty | Item | Code | Item Price £ | Total £ |
|---|---|---|---|---|
| | Sinclair QL Computer | 6000 | 399.00 | |
| | QLUB membership (one year) | 6100 | 35.00 | |
| | Postage & packing (any order over £390) | 6999 | 7.95 | |

Please tick the appropriate box-

☐ I enclose a cheque made payable to Sinclair Research Ltd for £______ £______

☐ Please charge my Access/Barclaycard Trustcard Account No.

Signature______

Mr/Mrs/Miss

Address

COT404

☐ Please send me a chart comparing the Sinclair QL computer with a______

☐ Please send me a Sinclair QL brochure

**sinclair**

Sinclair Research Ltd, Stanhope Road, Camberley, Surrey, GU15 3PS.


Space Pilot
A 64K BLOCK BUSTER
COMMODORE 64
ANIROG
SPACE PILOT
Fight your way to become The Master of the Skies travel through time to overcome the enemies of the past and future. Engage in dog fights with the squadrons of world war one to the fighters of the year 2084. Five Screens of Breathtaking Scenic Graphics.
J.S. £7.95
ANIROG SOFTWARE LTD.,
Trade Enquiries:
29 WEST HILL DARTFORD KENT (0322) 92513/8
Mail Order:
8 HIGH STREET HORLEY SURREY
24 HOUR CREDIT CARD SALES HORLEY (02934) 6083
PAYMENT BY CHEQUE P.O. ACCESS/VISA
50p POST & PACKAGING

# GENIE COMMANDS

Andrew Howard

**If you've got a 16K or larger Video Genie and you want to start writing programs in structured BASIC, you need VGBAS1. It also contains some very useful routines that satisfy other needs.**

VGBAS1 is a machine code program written for a Video Genie with a minimum of 16K memory which adds 13 useful new general-purpose commands to BASIC, three of them encouraging structured programming. The program occupies just under 1.5K and loads into 6000-672E (hex). The extra portion of end code is for initialisation only. Upon running, it relocates itself to be below BASIC but above everything else — it can therefore be used with Level III, for example. When loading several programs, VGBAS1 should be the last program to be loaded and run.

All of the new commands may be typed in as if they were normal BASIC statements. The 13 new commands are:

- REPEAT
- UNTIL
- PROC
- DEFPROC
- ENDPROC
- WHILE
- ENDWHILE
- JUMP
- !
- LINPUT
- CALL
- CANCEL
- EXEC

The first seven are structured statements, but the next two are not. The two after that are general-purpose, and the final two are special commands.

## REPEAT-UNTIL

This has the following structure:

REPEAT
(Program section)
UNTIL expression = true

The program section is repeatedly executed until the expression in the UNTIL statement gives a TRUE (ie value of 1) value when evaluated. On completion of the loop, control is transferred to the next statement following UNTIL.
Format of REPEAT: REPEAT
Format of UNTIL: UNTIL expression
Example programs are given in Fig. 1.

Nested REPEAT-UNTILs are allowed, up to 10 levels. They work a little like FOR-NEXT loops: you cannot jump in or out without an error. Any attempt to attain an eleventh level of nesting will result in a "TOO MANY REPEATS" error. If an UNTIL is encountered that has not been initialised by a REPEAT then an "UNTIL WITHOUT REPEAT" error will occur.

## PROC, DEFPROC — ENDPROC

The program structure of these commands looks like this:

(program)
PROC TITLE
(rest of program)
DEFPROC TITLE
Subroutine "TITLE"
ENDPROC

This command works like GOSUB except that instead of line numbers, labels are specified — an obvious advantage. The subroutine or procedure is called by the PROC command followed by the procedure name. The BASIC program is then switched for a DEFPROC with the same name following it. When found, the program is executed from there onwards, until an ENDPROC is found, upon which control is returned to the statement following the PROC. Note that DEFPROC does nothing on its own — it is merely a subroutine-beginning 'marker'.

```
10 PRINT"PRESS ANY KEY TO CONTINUE": A$=INKEY$
20 REPEAT: UNTIL INKEY$<>""
30 .....


10 PRINT"FIND SUM OF NUMBERS": PRINT"TERMINATE WITH
ZERO": T=0
20 REPEAT
30 INPUT"ENTER NEXT NUMBER";N: T=T+N
40 UNTIL N=0: PRINT"TOTAL =";T
```
**Fig. 1**

The DEFPROC at the beginning of the procedure **must** be the first item in a program line, otherwise it will not be found. Other statements may follow it, or start on new lines.
Format of PROC: PROC procedure name
Format of DEFPROC: DEFPROC procedure name
Format of ENDPROC: ENDPROC
The procedure name includes every character up to a colon or the end of a line. Program examples are given in Figs. 2 and 3.

As stated earlier, all VGBAS1 commands may be typed in as normal BASIC commands with one exception: after THEN and ELSE a colon should precede the VGBAS1 command, as illustrated in the example of Fig. 3. This example also shows that leading blanks and control codes before the procedure name are ignored, ie:
340 PROC    DEMO
will find
2300 DEFPROCDEMO: . . . .
as if it were
340 PROCDEMO
Spaces may also be inserted before the procedure name in the DEFPROC statement, just to improve readability.

Nested PROCedure calls are allowed, up to 10 levels. Any attempt to exceed 10 levels will result in "TOO MANY PROCEDURES". If an ENDPROC is encountered that is not part of a subroutine, an "ENDPROC WITHOUT PROC" error will result

```
10 CLS: PRINT"ENTER A HEXADECIMAL NUMBER OF FOUR
DIGITS": PRINT
20 INPUT H$: PROCHEXDEC: PRINT"DECIMAL =";D: END
50 DEFPROCHEXDEC: REM ** HEX TO DEC CONVERSION
60 D=0: P=3: H=1: REPEAT: A=ASC(MID$(H$,H,1)):
A=A-48: A=A+(7*(A>9))
70 D=D+(A*16*P): H=H+1: P=P-1: REM ** IS POWER RAISE
FUNCTION
80 UNTIL P<0: ENDPROC
```
**Fig. 2**

```
10 CLS: REPEAT: PRINT"TYPE SOMETHING IN": INPUT S$
20 IF S$<>"STOP" THEN: PROCINVERT: PRINT"REVERSED":
PRINTR$
30 PRINT: UNTIL S$="STOP"
40 END
50 DEFPROCINVERT: R$="": FOR A=LEN(S$) TO 1 STEP -1
60 R$=R$+MID$(S$,A,1): NEXT A: ENDPROC
```
**Fig. 3**

```
6000            00010         ORG   6000H
6000 E3         00020 TEST    EX    (SP),HL
6001 7C         00030         LD    A,H
6002 FE1D       00040         CP    1DH
6004 2003       00050         JR    NZ,NOTEXC
6006 7D         00060         LD    A,L
6007 FE5B       00070         CP    5BH
6009 E3         00080 NOTEXC  EX    (SP),HL
600A 2023       00090         JR    NZ,VECTOR
600C 114F60     00100 R0      LD    DE,CMDTBL
600F 23         00110         INC   HL
6010 0EFF       00120         LD    C,255
6012 E5         00130         PUSH  HL
6013 E1         00140 CMDLP   POP   HL
6014 E5         00150         PUSH  HL
6015 0C         00160         INC   C
6016 1A         00170 CMDLP1  LD    A,(DE)
6017 E67F       00180         AND   7FH
6019 47         00190         LD    B,A
601A 7E         00200         LD    A,(HL)
601B E67F       00210         AND   7FH
601D B8         00220         CP    B
601E 2812       00230         JR    Z,CHRMTC
6020 1A         00240 SKIP    LD    A,(DE)
6021 B7         00250         OR    A
6022 FA2860     00260 R1      JP    M,NXTCMD
6025 13         00270         INC   DE
6026 18F8       00280         JR    SKIP
6028 13         00290 NXTCMD  INC   DE
6029 1A         00300         LD    A,(DE)
602A B7         00310         OR    A
602B 20E6       00320         JR    NZ,CMDLP
602D E1         00330         POP   HL
602E 2B         00340         DEC   HL
602F C3781D     00350 VECTOR  JP    1D78H
6032 1A         00360 CHRMTC  LD    A,(DE)
6033 B7         00370         OR    A
6034 FA3B60     00380 R2      JP    M,FOUND
6037 13         00390         INC   DE
6038 23         00400         INC   HL
6039 18DB       00410         JR    CMDLP1
603B 23         00420 FOUND   INC   HL
603C D1         00430         POP   DE
603D E3         00440         EX    (SP),HL
603E 69         00450         LD    L,C
603F 2600       00460         LD    H,0
6041 29         00470         ADD   HL,HL
6042 117A60     00480 R3      LD    DE,ADDRS
6045 19         00490         ADD   HL,DE
6046 5E         00500         LD    E,(HL)
6047 23         00510         INC   HL
6048 56         00520         LD    D,(HL)
6049 D5         00530         PUSH  DE
604A DDE1       00540         POP   IX
604C E1         00550         POP   HL
604D DDE9       00560         JP    (IX)
604F 52         00570 CMDTBL  DEFM  'REPEA'
6050 45
6051 50
6052 45
6053 41
6054 D4         00580         DEFB  54H+80H
6055 55         00590         DEFM  'UNTI'
6056 4E
6057 54
6058 49
6059 CC         00600         DEFB  4CH+80H
605A 43         00610         DEFM  'CANCE'
605B 41
605C 4E
605D 43
605E 45
605F CC         00620         DEFB  4CH+80H
6060 43         00630         DEFM  'CAL'
6061 41
6062 4C
6063 CC         00640         DEFB  4CH+80H
6064 A1         00650         DEFB  21H+80H
6065 4A         00660         DEFM  'JUM'
6066 55
6067 4D
6068 D0         00670         DEFB  50H+80H
6069 50         00680         DEFM  'PRO'
606A 52
606B 4F
606C C3         00690         DEFB  43H+80H
606D 80         00700         DEFB  80H
606E 57         00710         DEFM  'WHIL'
606F 48
6070 49
6071 4C
6072 C5         00720         DEFB  45H+80H
6073 45         00730         DEFM  'EXE'
6074 58
6075 45
6076 C3         00740         DEFB  43H+80H
6077 4C         00750         DEFM  'L'
6078 89         00760         DEFB  89H
6079 00         00770         NOP
607A BA60       00780 ADDRS   DEFW  REPEAT
607C 5D61       00790         DEFW  UNTIL
607E 3F61       00800         DEFW  CANCEL
6080 AD62       00810         DEFW  CALL
6082 0663       00820         DEFW  DEFLAB
6084 6F63       00830         DEFW  JUMP
6086 CA63       00840         DEFW  PROC
6088 0964       00850         DEFW  END
608A 8564       00860         DEFW  WHILE
608C CF62       00870         DEFW  EXEC
608E 9A61       00880         DEFW  LINPUT
6090 9260       00890 REPPNT  DEFW  REPTBL
6092            00900 REPTBL  DEFS  40
60BA E5         00910 REPEAT  PUSH  HL
60BB 2A9060     00920 R4      LD    HL,(REPPNT)
60BE 11BA60     00930 R5      LD    DE,REPEAT
60C1 DF         00940         RST   18H
60C2 2006       00950         JR    NZ,REP2
60C4 E1         00960         POP   HL
60C5 3E01       00970         LD    A,1
60C7 C3C661     00980 R6      JP    ERROR
60CA E1         00990 REP2    POP   HL
60CB E5         01000         PUSH  HL
60CC DD2A9060   01010 R7      LD    IX,(REPPNT)
60D0 DD7500     01020         LD    (IX+0),L
60D3 DD7401     01030         LD    (IX+1),H
60D6 2AA240     01040         LD    HL,(40A2H)
60D9 DD7502     01050         LD    (IX+2),L
60DC DD7403     01060         LD    (IX+3),H
60DF 110400     01070         LD    DE,4
60E2 DD19       01080         ADD   IX,DE
60E4 DD229060   01090 R8      LD    (REPPNT),IX
60E8 E1         01100         POP   HL
60E9 C31E1D     01110         JP    1D1EH
60EC E5         01120 CANC0   PUSH  HL
60ED 2A9060     01130 R9      LD    HL,(REPPNT)
60F0 119260     01140 R10     LD    DE,REPTBL
60F3 DF         01150         RST   18H
60F4 2004       01160         JR    NZ,CAN2
60F6 E1         01170         POP   HL
60F7 C31E1D     01180         JP    1D1EH
60FA 2B         01190 CAN2    DEC   HL
60FB 2B         01200         DEC   HL
60FC 2B         01210         DEC   HL
60FD 2B         01220         DEC   HL
60FE 229060     01230 R11     LD    (REPPNT),HL
6101 E1         01240         POP   HL
6102 C31E1D     01250         JP    1D1EH
6105 E5         01260 CANC1   PUSH  HL
6106 2AA063     01270 R45     LD    HL,(PRCPNT)
6109 11A263     01280 R46     LD    DE,PRCTBL
610C DF         01290         RST   18H
610D 2004       01300         JR    NZ,CAN3
610F E1         01310         POP   HL
6110 C31E1D     01320         JP    1D1EH
6113 2B         01330 CAN3    DEC   HL
6114 2B         01340         DEC   HL
6115 2B         01350         DEC   HL
6116 2B         01360         DEC   HL
6117 22A063     01370 R47     LD    (PRCPNT),HL
611A E1         01380         POP   HL
611B C31E1D     01390         JP    1D1EH
611E            01400 TEMP    DEFS  2
6120 E5         01410 CANC2   PUSH  HL
6121 16FF       01420         LD    D,255
6123 221E61     01430 R61     LD    (TEMP),HL
6126 CD3619     01440         CALL  1936H
6129 F9         01450         LD    SP,HL
612A 22E840     01460         LD    (40E8H),HL
612D FE91       01470         CP    91H
612F 1E04       01480         LD    E,4
6131 C2A219     01490         JP    NZ,19A2H
6134 E1         01500         POP   HL
6135 211E1D     01510         LD    HL,1D1EH
6138 E3         01520         EX    (SP),HL
6139 2A1E61     01530 R60     LD    HL,(TEMP)
613C C3031F     01540         JP    1F03H
613F CD022B     01550 CANCEL  CALL  2B02H
6142 7B         01560         LD    A,E
6143 B7         01570         OR    A
6144 28A6       01580         JR    Z,CANC0
6146 FE01       01590         CP    1
6148 28BB       01600         JR    Z,CANC1
614A FE02       01610         CP    2
614C 28D2       01620         JR    Z,CANC2
614E FE03       01630         CP    3
6150 C29719     01640         JP    NZ,1997H
6153 E5         01650         PUSH  HL
6154 211E1D     01660         LD    HL,1D1EH
6157 E3         01670         EX    (SP),HL
6158 AF         01680 CANC3   XOR   A
6159 32CC62     01690 R69     LD    (EXECFG),A
615C C9         01700         RET
615D E5         01710 UNTIL   PUSH  HL
615E 2A9060     01720 R12     LD    HL,(REPPNT)
6161 119260     01730 R13     LD    DE,REPTBL
6164 DF         01740         RST   18H
6165 2006       01750         JR    NZ,UNT2
6167 E1         01760         POP   HL
6168 3E02       01770         LD    A,2
616A C3C661     01780 R14     JP    ERROR
616D E1         01790 UNT2    POP   HL
616E CD3723     01800         CALL  2337H
6171 E5         01810         PUSH  HL
6172 CD9409     01820         CALL  994H
6175 E1         01830         POP   HL
6176 C2EC60     01840 R71     JP    NZ,CANC0
6179 DD2A9060   01850 R15     LD    IX,(REPPNT)
617D DD2B       01860         DEC   IX
617F DD7E00     01870         LD    A,(IX+0)
6182 32A340     01880         LD    (40A3H),A
6185 DD2B       01890         DEC   IX
6187 DD7E00     01900         LD    A,(IX+0)
618A 32A240     01910         LD    (40A2H),A
618D DD2B       01920         DEC   IX
618F DD6600     01930         LD    H,(IX+0)
6192 DD2B       01940         DEC   IX
6194 DD6E00     01950         LD    L,(IX+0)
6197 C31E1D     01960         JP    1D1EH
619A E5         01970 LINPUT  PUSH  HL
619B 211E1D     01980         LD    HL,1D1EH
619E E3         01990         EX    (SP),HL
619F CD2828     02000         CALL  2828H
61A2 7E         02010         LD    A,(HL)
61A3 CDCD21     02020         CALL  21CDH
61A6 CD0D26     02030         CALL  260DH
61A9 CDF40A     02040         CALL  0AF4H
61AC D5         02050         PUSH  DE
61AD E5         02060         PUSH  HL
61AE CD6103     02070         CALL  361H
61B1 D1         02080         POP   DE
61B2 C1         02090         POP   BC
61B3 DABE1D     02100         JP    C,1DBEH
61B6 C5         02110         PUSH  BC
61B7 D5         02120         PUSH  DE
61B8 0600       02130         LD    B,0
61BA CD6828     02140         CALL  2868H
61BD E1         02150         POP   HL
61BE AF         02160         XOR   A
```

(the VGBAS1 equivalent of "RETURN WITHOUT GOSUB" error). Finally, if the PROC statement cannot find the specified procedure name, an "UNDEFINED LABEL" error will result.

## WHILE — ENDWHILE

This structure is of a more complex nature. Its basic structure is as follows:

WHILE expression = true
(execute program section)
ENDWHILE
(rest of program)

While the expression, when evaluated, has a TRUE value (ie if A is equal to 2, then the expression "A=2" is said to be TRUE), then the program section is executed. If the expression is FALSE, then the program section is skipped and control is transferred to the ENDWHILE statement. Note that ENDWHILE is like DEFPROC: it does nothing on its own and is merely an index. On a FALSE condition, the BASIC program is searched from the WHILE statement onwards for an ENDWHILE. If no ENDWHILE can be found after the WHILE, a "NO ENDWHILE STATEMENT" error will occur.

| | |
|---|---|
| Format of WHILE: | WHILE expression |
| Format of ENDWHILE: | ENDWHILE |

```
10 CLS: PRINT"FIND SUM OF NUMBERS. PRECEDE HEX
NUMBERS WITH '$', IE $41 (65 DECIMAL)": T=0: PRINT
"TERMINATE WITH ZERO": REPEAT
20 INPUT"NEXT NUMBER IS ";N$
30 WHILE LEFT$(N$,1)="$"
40 N$=RIGHT$(N$,LEN(N$)-1): N=0: P=LEN(N$)-1
50 FOR A=1 TO LEN(N$): B=ASC(MID$(N$,A,1))-48:
B=B+(7*(B>9))
60 N=N+(B*16^P): P=P-1: NEXT A: N$=STR$(N)
70 ENDWHILE: T=T+VAL(N$): UNTIL VAL(N$)=0: PRINT
"TOTAL =";T
```

**Fig. 4**

Figure 4 gives an example of the use of these words. It should be noted that a statement such as:

```
300 WHILE(A=4 AND H=8)
310 B(6)=0: B(A*K-1)=2/B(A*K-J): PROCALCOTHERS: A=0:
H=6
320 ENDWHILE: B(6)=B(6)+2: GOTO 258
```

can be specified as:

```
300 WHILE(A=4 AND H=8): B(6)=0: B(A*K-1)=B(A*K-J):
PROCALCOTHERS: A=0: H=6
320 ENDWHILE: B(6)=B(6)+2: GOTO 258
```

The above illustrates only the basic capabilities of WHILE. More practical structures can be constructed using multiple WHILEs:

WHILE expression = true
(Execute program section)
WHILE expression = true
(Execute program section)
. . .
Other WHILE statements
. . .
ENDWHILE

This structure is exactly the same as the previous one but with one major improvement. If the expression in a WHILE statement evaluates to FALSE, then the program section following it is skipped. From that WHILE statement onwards, the BASIC program is searched for the next WHILE or ENDWHILE. That is, on a FALSE condition the WHILE statement searches for an ENDWHILE. If, however, another WHILE is found instead of an ENDWHILE, the program from there onwards is executed. The format of the commands remains the same. Figure 5 gives an example.

Again, upon a FALSE condition, if no more WHILEs can be found and there is no ENDWHILE, a "NO ENDWHILE STATEMENT" message will be given.

```
100 CLS: INPUT"HOW MANY NUMBERS ";N: DIM NM(N): FOR
A=1 TO N: PRINT"ENTER VALUE NUMBER ";A;: INPUT NM(A):
NEXT A: CLS
110 PRINT"YOU HAVE FOUR COMMANDS:": PRINT: PRINT "A -
CALCULATE MEAN": PRINT "S - CALCULATE SUM": PRINT"Q -
CALCULATE SUM OF SQUARES": PRINT "E - END"
120 PRINT: PRINT"YOU CAN SPECIFY STRINGS OF COMMANDS,
FOR EXAMPLE AS OR QAE OR SIMPLY S": PRINT
130 INPUT"ENTER COMMAND(S) ";CM$: IF CM$="" THEN 130
140 FOR A=1 TO LEN(CM$): C$=MID$(CM$,A,1):
PROCCOMAND: NEXT A: RUN
200 DEFPROCCOMMAND: REM ** COMMAND EXECUTION
210 WHILE C$="A"
220 AM=0: FOR B=1 TO N: AM=AM+NM(B): NEXT B: PRINT
"ARITHMETIC MEAN =";AM/N
230 WHILE C$="S"
240 SM=0: FOR B=1 TO N: SM=SM+NM(B): NEXT B: PRINT
"SUM =";SM
250 WHILE C$="Q"
260 SQ=0: FOR B=1 TO N: SQ=SQ+NM(B)*NM(B): NEXT B:
PRINT"SUM OF SQUARES =";SQ
270 WHILE C$="E"
280 CLEAR 50: END
290 ENDWHILE: ENDPROC
```

**Fig. 5**

Actually, it would be better to change the RUN in line 140 of Fig. 5 to GOTO 130.

As you may have worked out, nested WHILEs are not allowed. To overcome this, the WHILE structure becomes even more complex. The new statement formats are as follows (items in square brackets are optional):

| | |
|---|---|
| Format of WHILE: | WHILE [(expression1),] expression 2 |
| Format of ENDWHILE: | ENDWHILE [(expression 1)] |

The structure is:

WHILE (expression 1), expression 2 = true
(Execute program section)
WHILE (expression 1), expression 2 = true
(Execute inner program section)
ENDWHILE (expression 1)
ENDWHILE (expression 1)

This mode of the WHILE structure can be very powerful if used well. Note that the 'normal' WHILE structure can still be used (normal being the structure as shown in Fig. 5): this new, more complex mode is optional.

When executed WHILE behaves as normal — upon a true condition the program section is executed as normal. However, upon a false condition, instead of searching for the next WHILE, the program is searched for the next WHILE with a matching value for expression 1. That is, if a WHILE is found with a differing value of expression 1 than that of the WHILE currently being executed, then it will be passed by and the next WHILE will be searched for. If, instead of a WHILE, an ENDWHILE is encountered with a matching value for expression 1, then control will be transferred here. As for WHILE, if an ENDWHILE with a differing value for expression 1 to that of the current WHILE is found, then it will be ignored.

The example above shows how this feature can be used in nesting WHILEs. As for DEFPROC, all ENDWHILEs and WHILEs must start on a new line:

456 PRINT: AG=0: ENDWHILE

is not allowed, but:

456 PRINT: AG=0
457 ENDWHILE

is allowed. An example of the new WHILE structure is given in Fig. 6. This is a rather silly example, but I couldn't think of anything else.

It should be noted that:
WHILE (expression 1), expression 2
where expression 1 evaluates to zero is exactly the same as:
WHILE expression
That is, on FALSE, WHILE(0), expression 2 would find the next WHILE(0), expression 2 or for the next WHILE expression or the next ENDWHILE(0) or the next ENDWHILE.

```
61BF C3321F   02170          JP     1F32H
61C2 00       02180 ERR      NOP
61C3          02190 ERREXT   DEFS   3
61C6 32C261   02200 ERROR    LD     (ERR),A
61C9 3AA641   02210          LD     A,(41A6H)
61CC 2AA741   02220          LD     HL,(41A7H)
61CF 32C361   02230 R16      LD     (ERREXT),A
61D2 22C461   02240 R17      LD     (ERREXT+1),HL
61D5 3EC3     02250          LD     A,195
61D7 21E361   02260 R18      LD     HL,ERROR2
61DA 32A641   02270          LD     (41A6H),A
61DD 22A741   02280          LD     (41A7H),HL
61E0 C39719   02290          JP     1997H
61E3 E1       02300 ERROR2   POP    HL
61E4 3AC361   02310 R19      LD     A,(ERREXT)
61E7 2AC461   02320 R20      LD     HL,(ERREXT+1)
61EA 32A641   02330          LD     (41A6H),A
61ED 22A741   02340          LD     (41A7H),HL
61F0 3AC261   02350 R21      LD     A,(ERR)
61F3 87       02360          ADD    A,A
61F4 4F       02370          LD     C,A
61F5 0600     02380          LD     B,0
61F7 210862   02390 R22      LD     HL,ERRTBL-2
61FA 09       02400          ADD    HL,BC
61FB 5F       02410          LD     E,(HL)
61FC 23       02420          INC    HL
61FD 56       02430          LD     D,(HL)
61FE EB       02440          EX     DE,HL
61FF 3E3F     02450          LD     A,'?'
6201 CD3300   02460          CALL   33H
6204 CD752B   02470          CALL   2B75H
6207 C3FE19   02480          JP     19FEH
620A 1662     02490 ERRTBL   DEFW   MANYRP
620C 2762     02500          DEFW   NOREP
620E 3C62     02510          DEFW   NOLAB
6210 4C62     02520          DEFW   MANYPC
6212 6062     02530          DEFW   NOPRC
6214 7562     02540          DEFW   NOENDW
6216 54       02550 MANYRP   DEFM   'TOO MANY REPEATS'
6217 4F
6218 4F
6219 20
621A 4D
621B 41
621C 4E
621D 59
621E 20
621F 52
6220 45
6221 50
6222 45
6223 41
6224 54
6225 53
6226 00       02560          NOP
6227 55       02570 NOREP    DEFM   'UNTIL WITHOUT REPEAT'
6228 4E
6229 54
622A 49
622B 4C
622C 20
622D 57
622E 49
622F 54
6230 48
6231 4F
6232 55
6233 54
6234 20
6235 52
6236 45
6237 50
6238 45
6239 41
623A 54
623B 00       02580          NOP
623C 55       02590 NOLAB    DEFM   'UNDEFINED LABEL'
623D 4E
623E 44
623F 45
6240 46
6241 49
6242 4E
6243 45
6244 44
6245 20
6246 4C
6247 41
6248 42
6249 45
624A 4C
624B 00       02600          NOP
624C 54       02610 MANYPC   DEFM   'TOO MANY PROCEDURES'
624D 4F
624E 4F
624F 20
6250 4D
6251 41
6252 4E
6253 59
6254 20
6255 50
6256 52
6257 4F
6258 43
6259 45
625A 44
625B 55
625C 52
625D 45
625E 53
625F 00       02620          NOP
6260 45       02630 NOPRC    DEFM   'ENDPROC WITHOUT PROC'
6261 4E
6262 44
6263 50
6264 52
6265 4F
6266 43
6267 20
6268 57
6269 49
626A 54
626B 48
626C 4F
626D 55
626E 54
626F 20
6270 50
6271 52
6272 4F
6273 43
6274 00       02640          NOP
6275 4E       02650 NOENDW   DEFM   'NO ENDWHILE STATEMENT'
6276 4F
6277 20
6278 45
6279 4E
627A 44
627B 57
627C 48
627D 49
627E 4C
627F 45
6280 20
6281 53
6282 54
6283 41
6284 54
6285 45
6286 4D
6287 45
6288 4E
6289 54
628A 00       02660          NOP
628B E3       02670 NEWRUN   EX     (SP),HL
628C 7C       02680          LD     A,H
628D FE1B     02690          CP     1BH
628F 2003     02700          JR     NZ,NOTRUN
6291 7D       02710          LD     A,L
6292 FE8F     02720          CP     8FH
6294 E3       02730 NOTRUN   EX     (SP),HL
6295 2803     02740          JR     Z,RUN2
6297 C30000   02750 RUNEXT   JP     0
629A E5       02760 RUN2     PUSH   HL
629B 219260   02770 R23      LD     HL,REPTBL
629E 229060   02780 R24      LD     (REPPNT),HL
62A1 21A263   02790 R42      LD     HL,PRCTBL
62A4 22A063   02800 R43      LD     (PRCPNT),HL
62A7 CD5861   02810 R70      CALL   CANC3
62AA E1       02820          POP    HL
62AB 18EA     02830          JR     RUNEXT
62AD 2B       02840 CALL     DEC    HL
62AE D7       02850          RST    10H
62AF FE24     02860          CP     '$'
62B1 2009     02870          JR     NZ,EXP
62B3 D7       02880          RST    10H
62B4 D29719   02890          JP     NC,1997H
62B7 CD5A1E   02900          CALL   1E5AH
62BA 1809     02910          JR     CALL2
62BC CD3723   02920 EXP      CALL   2337H
62BF E5       02930          PUSH   HL
62C0 CD7F0A   02940          CALL   0A7FH
62C3 EB       02950          EX     DE,HL
62C4 E1       02960          POP    HL
62C5 E5       02970 CALL2    PUSH   HL
62C6 211E1D   02980          LD     HL,1D1EH
62C9 E3       02990          EX     (SP),HL
62CA D5       03000          PUSH   DE
62CB C9       03010          RET
62CC 00       03020 EXECFG   NOP
62CD          03030 EXECST   DEFS   2
62CF CD3523   03040 EXEC     CALL   2335H
62D2 CF       03050          RST    8
62D3 29       03060          DEFM   ')'
62D4 3E01     03070          LD     A,1
62D6 32CC62   03080 R63      LD     (EXECFG),A
62D9 22CD62   03090 R64      LD     (EXECST),HL
62DC CDD729   03100          CALL   29D7H
62DF 4E       03110          LD     C,(HL)
62E0 23       03120          INC    HL
62E1 5E       03130          LD     E,(HL)
62E2 23       03140          INC    HL
62E3 56       03150          LD     D,(HL)
62E4 0600     03160          LD     B,0
62E6 78       03170          LD     A,B
62E7 B1       03180          OR     C
62E8 CA9719   03190          JP     Z,1997H
62EB 2AA740   03200          LD     HL,(40A7H)
62EE E5       03210          PUSH   HL
62EF C5       03220          PUSH   BC
62F0 EB       03230          EX     DE,HL
62F1 EDB0     03240          LDIR
62F3 C1       03250          POP    BC
62F4 E1       03260          POP    HL
62F5 E5       03270          PUSH   HL
62F6 09       03280          ADD    HL,BC
62F7 363A     03290          LD     (HL),3AH
62F9 23       03300          INC    HL
62FA 3680     03310          LD     (HL),80H
62FC 23       03320          INC    HL
62FD 3600     03330          LD     (HL),0
62FF E1       03340          POP    HL
6300 CDC01B   03350          CALL   1BC0H
6303 C31E1D   03360          JP     1D1EH
6306 013A3A   03370 DEFLAB   LD     BC,3A3AH
6309 CD0B1F   03380          CALL   1F0BH
630C C31E1D   03390          JP     1D1EH
630F 00       03400 FNDCMD   NOP
6310 2B       03410 FINDLB   DEC    HL
6311 D7       03420          RST    10H
6312 4D       03430          LD     C,L
6313 44       03440          LD     B,H
6314 2AA440   03450          LD     HL,(40A4H)
6317 7A       03460          LD     A,D
6318 320F63   03470 R31      LD     (FNDCMD),A
631B 7E       03480 JPLOOP   LD     A,(HL)
631C 23       03490          INC    HL
```

```
10 CLS: PRINT"ENTER A TYPE OF ANIMAL": PRINT"MAKE IT
A CAT, DOG, OR MOUSE": REPEAT: INPUT"ANIMAL ";A$:
UNTIL A$="CAT" OR A$="DOG" OR A$="MOUSE"
20 INPUT"ENTER THE NAME OF A FOOD FOR THIS ANIMAL ";
F$
30 REM ** F$ SHOULD BE "STOP" TO STOP THE PROGRAM
40 WHILE F$<>"STOP"
50 PROC PRINTDETAILS: PROCDELAY: RUN
60 ENDWHILE: END

100 DEFPROC   DELAY: REM ** PROCEDURES
110 FOR A=1 TO 1000: NEXT A: ENDPROC

200 DEFPROC   PRINTDETAILS
210 WHILE(1),A$="DOG"
220 PRINT"A DOG IS MAN'S BEST FRIEND"
230 WHILE(2),F$="MEAT"
240 PRINT"DOESN'T YOUR DOG DESERVE MEAT?"
250 ENDWHILE(2): PRINT"DOGS BARK LOUDLY"
260 WHILE(1),A$="MOUSE"
270 PRINT"SOME PEOPLE ARE SCARED OF MICE"
280 PRINT"BUT CATS AREN'T"
290 WHILE(1),A$="CAT"
300 PRINT"CATS ARE WIDELY LOVED AS PETS":PRINT"THOUGH
NOT BY CT'S EDITOR": PRINT"DO YOU LIKE CATS ";
310 REPEAT: REPEAT: C$="": INPUT C$: UNTIL C$<>"":
C$=LEFT$(C$,1): UNTIL C$="Y" OR C$="N"
320 WHILE(2),F$="FISH"
330 WHILE(3),C$="Y"
340 PRINT"YOUR CAT LIKES FISH. YOU LIKE YOUR CAT."
350 PRINT"DO YOU LIKE FISH?"
360 ENDWHILE(3): PRINT"FISH SWIM . . ."
370 WHILE(2),F$="DOG FOOD"
380 PRINT"STRANGE CAT! DOES IT BARK ??"
390 ENDWHILE(2): PRINT"FISH HATE CATS."
400 ENDWHILE(1): ENDPROC
```

**Fig. 6**

In the listing of Fig. 6, ordinary numbers (constants) were used for expression 1. Variables may also be used, hence the notation 'expression 1':

```
560 WHILE(A*B+(INT(SIN(.75+B+J)/.05)+.6+PZ(J)),
expression 2

235 WHILE(PEEK(32745)),expression 2

65  WHILE(A),expression 2
```

and so on. The result of any expression 1 will be converted to integer form and must be between 0 and 255 inclusive.

But the really powerful feature of the VGBAS1 WHILE structure is the ability to change the program structure by changing the result of expression 1. For example, if in line 290 of Fig. 6, instead of WHILE(1), A$="CAT", there was WHILE(J), A$="CAT", and the variable J had been set to 1 at line 10, for example, then changing J to 2 would change the whole structure of the PRINTDETAILS procedure. That is, if J was 2, then upon a false condition in line 260, line 290 would be passed by because it had a different value for expression 1. This example is only a simple one — used properly, WHILE can provide very complex programs.

```
10 REM ** INITIALISE VARIABLES
20 ...

70 JUMPTITLES: REM ** GO AND DO TITLE AND START
PROGRAM
. . . REST OF PROGRAM . . .

900 !TITLES: CLS: PRINT" . . .


1000 ! LOOP: CLS: INPUT"TYPE ANY NUMBER ";N
1002 WHILE N=1
1004 PRINT"NUMBER ONE": PROC SUBONE
1006 WHILE N=2
1008 PRINT"NUMBER TWO": PROC SUBTWO

. . . OTHER WHILES . . .

1070 REM ** LOOP AROUND AND GET ANOTHER NUMBER
1080 FOR A=1 TO 1000: NEXT A: JUMPLOOP
```

**Fig. 7**

## JUMP, !

This command is a new form of GOTO, in that no line numbers have to be specified:

| | |
|---|---|
| Format of JUMP: | JUMP label |
| Format of ! : | ! label |

The ! command is like DEFPROC: it marks the start of a section of program. It must be at the start of a program line. The jump command transfers control to the ! with a matching label to that of the JUMP. The label does not include leading blanks and control codes, but does include every character from there onwards to the next colon or end of the line. Figure 7 shows some examples of JUMP.

If the JUMP label cannot be found within the program, then "UNDEFINED LABEL" will result. Note that the following are not allowed:

```
560 PROCGETKEY: . . . rest of program
700 !GETKEY: . . . rest of program
```

and

```
2150 DEFPROCPRINTTITLE: . . .
3000 JUMP PRINTTITLE
```

## LINPUT

This command allows a string literal to be input, including commas. The format is:
LINPUT ["message";] string variable
where the items in square brackets are optional. This command is exactly the same as an ordinary input except that no ? prompt is printed, commas and colons are allowed, and it can only be used with string variables. Figure 8 contains two examples: in the first case, when the program is RUN, CM$ will contain the **whole** string typed, including commas and colons. The second program is followed by a sample run.

```
10 CLS: PRINT"ENTER COMMAND": PRINT
20 PRINT"READY": CM$="": REPEAT: LINPUT CM$: UNTIL
CM$<>""
. . . rest of program . . .

10 CLS: PRINT"PRESONAL DETAILS": PRINT
20 LINPUT"ENTER YOUR NAME AND TELEPHONE NUMBER";PD$
30 PRINT: PRINT PD$

PERSONAL DETAILS

ENTER YOUR NAME AND TELEPHONE NUMBER:FRED BLOGGS,
01-636-7834

FRED BLOGGS, 01-636-7834
```

**Fig. 8**

## CALL

This command has two formats:
CALL expression
CALL $absolute address
It is used to transfer control to a machine code routine. On entry to this routine, the HL register pair will contain the address of the next byte of the BASIC program to be executed. A RETurn instruction will return control to BASIC at the point left off.

In the CALL expression format, expression must be supplied as for a POKE or PEEK. That is, to call a machine code subroutine at 28672 (7000 hex) would require CALL 28672, a CALL to 32767 would require CALL 32767, and so on. However, to call 32768 would require CALL−32768, 61440 would require CALL −4096, and so on. Therefore expression must be supplied as an integer, that is if the address is greater than 32767 then subtract 65536. Expression can be any expression, and therefore the CALL command can be flexible. For example, a menu program in BASIC that executes the appropriate machine code subroutine according to the user's option is given in Fig. 9.

The CALL dollar sign format is not as flexible. It is absolute, ie the address is not evaluated — it is merely converted to binary. That means that variables cannot be used. However, it is

```
631D B6                 OR      (HL)
631E 2816               JR      Z,NOLABL
6320 E5                 PUSH    HL
6321 23                 INC     HL
6322 5E                 LD      E,(HL)
6323 23                 INC     HL
6324 56                 LD      D,(HL)
6325 D7                 RST     10H
6326 C5                 PUSH    BC
6327 47                 LD      B,A
6328 3A0F63   R32       LD      A,(FNDCMD)
632B B8                 CP      B
632C C1                 POP     BC
632D 280C               JR      Z,LABEL
632F E1       NOLBMT    POP     HL
6330 56                 LD      D,(HL)
6331 2B                 DEC     HL
6332 5E                 LD      E,(HL)
6333 EB                 EX      DE,HL
6334 18E5               JR      JPLOOP
6336 3E03     NOLABL    LD      A,3
6338 C3C661   R25       JP      ERROR
633B 23       LABEL     INC     HL
633C E5                 PUSH    HL
633D C5                 PUSH    BC
633E CD7F63   R34       CALL    TSTPRC
6341 C1                 POP     BC
6342 3803               JR      C,PRCDEF
6344 E1                 POP     HL
6345 1801               JR      LABEL2
6347 F1       PRCDEF    POP     AF
6348 2B       LABEL2    DEC     HL
6349 D7                 RST     10H
634A E5                 PUSH    HL
634B C5                 PUSH    BC
634C 0A       CPLABL    LD      A,(BC)
634D BE                 CP      (HL)
634E 2804               JR      Z,LABMTC
6350 C1                 POP     BC
6351 E1                 POP     HL
6352 18DB               JR      NOLBMT
6354 23       LABMTC    INC     HL
6355 03                 INC     BC
6356 7E                 LD      A,(HL)
6357 B7                 OR      A
6358 2804               JR      Z,LBMTC1
635A FE3A               CP      3AH
635C 20EE               JR      NZ,CPLABL
635E 0A       LBMTC1    LD      A,(BC)
635F B7                 OR      A
6360 2804               JR      Z,LBMTC2
6362 FE3A               CP      3AH
6364 20E6               JR      NZ,CPLABL
6366 F1       LBMTC2    POP     AF
6367 F1                 POP     AF
6368 E3                 EX      (SP),HL
6369 23                 INC     HL
636A 5E                 LD      E,(HL)
636B 23                 INC     HL
636C 56                 LD      D,(HL)
636D E1                 POP     HL
636E C9                 RET
636F 1621     JUMP      LD      D,'!'
6371 CD1063   R39       CALL    FINDLB
6374 ED53A240           LD      (40A2H),DE
6378 C31E1D             JP      1D1EH
637B 50       MSPROC    DEFM    'PROC'
637C 52
637D 4F
637E 43
637F 117B63   TSTPRC    LD      DE,MSPROC
6382 0604               LD      B,4
6384 1A       TSTLP     LD      A,(DE)
6385 BE                 CP      (HL)
6386 2802               JR      Z,TSTLPM
6388 AF                 XOR     A
6389 C9                 RET
638A 13       TSTLPM    INC     DE
638B 23                 INC     HL
638C 10F6               DJNZ    TSTLP
638E 37                 SCF
638F C9                 RET
6390 E5       DEF       PUSH    HL
6391 CD7F63   R33       CALL    TSTPRC
6394 3806               JR      C,DEFPRC
6396 E1                 POP     HL
6397 2B                 DEC     HL
6398 D7                 RST     10H
6399 C30000   DEFEXT    JP      0
639C D1       DEFPRC    POP     DE
639D C30663   R44       JP      DEFLAB
63A0 A263     PRCPNT    DEFW    PRCTBL
63A2          PRCTBL    DEFS    40
63CA 16B0     PROC      LD      D,0B0H
63CC CD1063   R56       CALL    FINDLB
63CF E5                 PUSH    HL
63D0 D5                 PUSH    DE
63D1 2AA063   R36       LD      HL,(PRCPNT)
63D4 11CA63   R37       LD      DE,PROC
63D7 DF                 RST     18H
63D8 2007               JR      NZ,PROC2
63DA D1                 POP     DE
63DB E1                 POP     HL
63DC 3E04               LD      A,4
63DE C3C661   R38       JP      ERROR
63E1 D1       PROC2     POP     DE
63E2 DD2AA063 R40       LD      IX,(PRCPNT)
63E6 DD7100             LD      (IX+0),C
63E9 DD23               INC     IX
63EB DD7000             LD      (IX+0),B
63EE DD23               INC     IX
63F0 2AA240             LD      HL,(40A2H)
63F3 DD7500             LD      (IX+0),L
63F6 DD23               INC     IX
63F8 DD7400             LD      (IX+0),H
63FB DD23               INC     IX
63FD DD22A063 R41       LD      (PRCPNT),IX
6401 E1                 POP     HL
6402 ED53A240           LD      (40A2H),DE
```

```
6406 C31E1D             JP      1D1EH
6409 2B       END       DEC     HL
640A D7                 RST     10H
640B 201E               JR      NZ,ENDCMD
640D 3ACC62   R65       LD      A,(EXECFG)
6410 B7                 OR      A
6411 280B               JR      Z,ENDNRM
6413 AF                 XOR     A
6414 32CC62   R66       LD      (EXECFG),A
6417 E1                 POP     HL
6418 2ACD62   R67       LD      HL,(EXECST)
641B C31E1D             JP      1D1EH
641E E5       ENDNRM    PUSH    HL
641F 211E1D             LD      HL,1D1EH
6422 E3                 EX      (SP),HL
6423 C3AF1D             JP      1DAFH
6426 57       MSWHIL    DEFM    'WHILE'
6427 48
6428 49
6429 4C
642A 45
642B E5       ENDCMD    PUSH    HL
642C CD7F63   R48       CALL    TSTPRC
642F 302C               JR      NC,ENDWHL
6431 E3                 EX      (SP),HL
6432 2AA063   R51       LD      HL,(PRCPNT)
6435 11A263   R52       LD      DE,PRCTBL
6438 DF                 RST     18H
6439 2006               JR      NZ,ENDPRC
643B E1                 POP     HL
643C 3E05               LD      A,5
643E C3C661   R50       JP      ERROR
6441 E1       ENDPRC    POP     HL
6442 2AA063   R53       LD      HL,(PRCPNT)
6445 2B                 DEC     HL
6446 2B                 DEC     HL
6447 2B                 DEC     HL
6448 2B                 DEC     HL
6449 22A063   R68       LD      (PRCPNT),HL
644C 5E                 LD      E,(HL)
644D 23                 INC     HL
644E 56                 LD      D,(HL)
644F 23                 INC     HL
6450 7E                 LD      A,(HL)
6451 32A240             LD      (40A2H),A
6454 23                 INC     HL
6455 7E                 LD      A,(HL)
6456 32A340             LD      (40A3H),A
6459 EB                 EX      DE,HL
645A C31E1D             JP      1D1EH
645D E1       ENDWHL    POP     HL
645E 112664   R54       LD      DE,MSWHIL
6461 0605               LD      B,5
6463 CD8463   R55       CALL    TSTPRC+5
6466 D29719             JP      NC,1997H
6469 C30663   R72       JP      DEFLAB
646C 00       WHLNO     NOP
646D 2B       TSTWNO    DEC     HL
646E D7                 RST     10H
646F FE28               CP      '('
6471 C0                 RET     NZ
6472 D7                 RST     10H
6473 CD3723             CALL    2337H
6476 CD052B             CALL    2B05H
6479 C2B207             JP      NZ,7B2H
647C D5                 PUSH    DE
647D 2B                 DEC     HL
647E D7                 RST     10H
647F CF                 RST     8
6480 29                 DEFM    ')'
6481 D1                 POP     DE
6482 AF                 XOR     A
6483 7B                 LD      A,E
6484 C9                 RET
6485 AF       WHILE     XOR     A
6486 326C64   R74       LD      (WHLNO),A
6489 CD6D64   R75       CALL    TSTWNO
648C 2005               JR      NZ,WHILE2
648E 326C64   R73       LD      (WHLNO),A
6491 CF                 RST     8
6492 2C                 DEFM    ','
6493 CD3723   WHILE2    CALL    2337H
6496 E5                 PUSH    HL
6497 CD9409             CALL    994H
649A E1                 POP     HL
649B C21E1D             JP      NZ,1D1EH
649E CD071F             CALL    1F07H
64A1 23                 INC     HL
64A2 E5       SRCHWL    PUSH    HL
64A3 7E                 LD      A,(HL)
64A4 23                 INC     HL
64A5 B6                 OR      (HL)
64A6 3E06               LD      A,6
64A8 CAC661             JP      Z,ERROR
64AB 23                 INC     HL
64AC 23                 INC     HL
64AD E5                 PUSH    HL
64AE D7                 RST     10H
64AF FE80               CP      80H
64B1 2001               JR      NZ,SKPEND
64B3 D7                 RST     10H
64B4 112664   SKPEND    LD      DE,MSWHIL
64B7 0605               LD      B,5
64B9 CD8463   R62       CALL    TSTPRC+5
64BC 302F               JR      NC,NOWHIL
64BE 3A6C64   R77       LD      A,(WHLNO)
64C1 B7                 OR      A
64C2 280B               JR      Z,NOWLNO
64C4 CD6D64   R76       CALL    TSTWNO
64C7 2024               JR      NZ,NOWHIL
64C9 3A6C64   R78       LD      A,(WHLNO)
64CC BB                 CP      E
64CD 201E               JR      NZ,NOWHIL
64CF E1       NOWLNO    POP     HL
64D0 E1                 POP     HL
64D1 23                 INC     HL
64D2 23                 INC     HL
64D3 5E                 LD      E,(HL)
64D4 23                 INC     HL
```

```
10 DIM MS(5): FOR A=1 TO 5: READ MS(A): NEXT A: REM
** READ M/C SUBROUTINE ADDRESSES
20 DATA 28672,29254,29998,30327,30600
30 !MENU: CLS: PRINT,"    MASTER MENU": PRINT
40 PRINT,"(1) . . . "
50 PRINT,"(2) . . . "
60 PRINT,"(3) . . . "
70 PRINT,"(4) . . . "
80 PRINT,"(5) . . . "
90 PRINT,"(6) EXIT PROGRAM": PRINT
100 UU=0: REPEAT: PRINT,"    YOUR OPTION ";: INPUT
UU: UNTIL UU>0 AND UU<7 AND UU=INT(UU)
110 WHILE UU<6
120 CALL MS(UU): FOR A=1 TO 1000: NEXT A: JUMPMENU
130 ENDWHILE: END
```

**Fig. 9**

available because addresses do not have to be specified in 'integer' form, so 61440 can be specified as CALL$61440.

The CALL command can be used to add new commands to BASIC, as in the program of Fig. 10 which adds a BEEP command. This routine is executed by:

CALL 32512, duration, pitch   or
CALL $32512, duration, pitch
where duration and pitch must be less than 256. Examples are:

```
350 DEFPROCERROR: REM ** ERROR ROUTINE
360 PRINT"INPUT ERROR": CALL 32512,50,50: REM **
SIGNAL ERROR
370 ENDPROC
```

## CANCEL

Cancel is a general 'abort' command and is used to abandon various operations.
Format: CANCEL expression
Expression, when evaluated, must result in one of four values:

| Result of expression | Action taken |
|---|---|
| 0 | Cancel current REPEAT-UNTIL loop |
| 1 | Cancel current PROC call |
| 2 | Cancel current GOSUB |
| 3 | Cancel EXEC mode |

IF a CANCEL2 is specified, that will cancel the RETURN address, so that a future RETURN will produce an RG error. That is, the outcome of a CANCEL 2 would be as if the calling program GOTOed instead of GOSUBed.

The same applies to CANCEL 1. It will cancel the ENDPROC address so any future ENDPROC will produce an "ENDPROC WITHOUT PROC" error.

```
                ORG    7F00H
7F00 CF         RST    8          ;check for comma
7F01 2C         DEFB   ','        ;
7F02 CD 21 7F   CALL   7F21H      ;evaluate duration
7F05 43         LD     B,E        ;B = duration
7F06 C5         PUSH   BC         ;
7F07 CF         RST    8          ;check for comma
7F08 2C         DEFB   ','        ;
7F09 CD 21 7F   CALL   7F21H      ;evaluate pitch
7F0C C1         POP    BC         ;
7F0D 4B         LD     C,E        ;C = pitch
7F0E C5         PUSH   BC         ;
7F0F 3E 02      LD     A,2        ;first half of sound
                                  ;blip
7F11 41         LD     B,C        ;B = pitch
7F12 D3 FF      OUT    (255),A    ;
7F14 10 FE      DJNZ   7F14H      ;wait a bit
7F16 3E 01      LD     A,1        ;second half of blip
7F18 41         LD     B,C        ;
7F19 D3 FF      OUT    (255),A    ;
7F1B 10 FE      DJNZ   7F1BH      ;
7F1D C1         POP    BC         ;
7F1E 10 EE      DJNZ   7F0EH      ;loop until dur = 0
7F20 C9         RET               ;return to BASIC
7F21 CD 02 2B   CALL   2B02H      ;evaluate expression
7F24 C8         RET    Z          ;return if result OK
7F25 C3 B2 07   JP     07B2H      ;overflow if result>255
7F28            END
```

**Fig. 10**

If a program jumps out of a REPEAT-UNTIL loop, CANCEL0 will cancel the incomplete loop. CANCEL3 will abort EXEC mode (explained later).

If expression, when evaluated, results in a value less than zero or greater than three, a syntax error will occor.

## EXEC

The EXEC command is a new type of command and is used to execute a string, just as if it was a line of BASIC.
Format: EXEC (string expression)
The string is tokenised and an END command is placed on the end. The tokenised string is then executed by the BASIC interpreter. When the END command is encountered, control is returned to the statement following the EXEC command. Note that nested EXECs are not allowed.

When EXEC is executed, EXEC mode comes into force. The END command exits EXEC mode. It is possible for the string to contain statements which transfer control to the areas of the program. This is perfectly okay, but must be used with care. If this feature is used, then END will return control to the statement following the EXEC command. This is where CANCEL3 is useful. If, via EXEC, control is transferred to another area of the program, then CANCEL3 will cancel EXEC mode, and the END command will act as normal, ending the program.

```
10 CLEAR 1000: READ NC: DIM SC$(NC): FOR A=1 TO NC:
READ SC$(A): NEXT A
20 REM ** ARRAY SC$ CONTAINS ALL COMMANDS
30 DATA 5,SAVE,LOAD,VERIFY,PRINT,EDIT
40 CLS: PRINT"WORD PROCESSOR": PRINT"COMMAND MODE":
PRINT
50 PRINT"COMMAND ?": CM$="": REPEAT: LINPUT"*";CM$:
UNTIL CM$<>""
60 REM ** TEST FOR COMMAND AND EXECUTE APPROPRIATE
PROCEDURE
70 A=1: REPEAT
80 WHILE CM$=SC$(A): CANCEL0: EXEC("PROC"+SC$(A)):
GOTO 50
90 ENDWHILE: A=A+1: UNTIL A=6: GOTO 40

100 REM ** PROCEDURES
110 DEFPROCLOAD: . . . load routine
200 DEFPROCSAVE: . . . save routine
. . . rest of procedures . . .
```

**Fig. 11**

The EXEC command must be used with care. The only restriction is that INPUT and LINPUT cannot be used without some kind of error resulting, and EXEC cannot be used in FOR-NEXT loops. Examples of EXEC are given in Fig. 11.

The EXEC command can also be used to evaluate expressions, for example:

```
A$="A=B*A+C+.76+INT(COS(A)*.75*SIN(B)+(C/A+A))":
EXEC(A$)
```

The program in Fig. 11 shows that EXEC can be a very useful command. However, it also shows that care must be taken when using procedure names and labels. It should work okay when the user types VERIFY, PRINT or EDIT. However, if either SAVE or LOAD is typed, the machine will respond with a syntax error in line 80. This is because the string to be executed is either PROCSAVE or PROCLOAD. Note that PROC SAVE and PROC LOAD work. Why? -Because there is a hidden (embedded) command in the original examples — CSAVE and CLOAD. To overcome this, change line 80 to:

```
80 WHILE CM$=SC$(A): CANCEL0: EXEC("PROC "+SC$(A)):
GOTO 50
```

The DEFPROCs in lines 110 and 200 will also have to be changed:

```
110 DEFPROC LOAD: . . .
200 DEFPROC SAVE: . . .
```

The program should now work fully. If you are thinking "Well, he probably typed out the program, then typed it into his computer and found it didn't work. So he had to add a bit to his instructions" — you're right!!

That concludes the instructions for VGBAS1.

```
64D5 56       05810          LD     D,(HL)
64D6 ED53A240 05820          LD     (40A2H),DE
64DA 23       05830          INC    HL
64DB CD5803   05840          CALL   358H
64DE B7       05850          OR     A
64DF C4A01D   05860          CALL   NZ,1DA0H
64E2 22E640   05870          LD     (40E6H),HL
64E5 ED73E840 05880          LD     (40E8H),SP
64E9 2B       05890          DEC    HL
64EA C35A1D   05900          JP     1D5AH
64ED E1       05910 NOWHIL   POP    HL
64EE E1       05920          POP    HL
64EF 5E       05930          LD     E,(HL)
64F0 23       05940          INC    HL
64F1 56       05950          LD     D,(HL)
64F2 EB       05960          EX     DE,HL
64F3 18AD     05970          JR     SRCHWL
64F5 00       05980          NOP
64F6 56       05990 TITLE    DEFM   'VGBAS1 VIDEO GENIE LEVEL 2 INTERP
RETER EXTENSION 1'
64F7 47
64F8 42
64F9 41
64FA 53
64FB 31
64FC 20
64FD 56
64FE 49
64FF 44
6500 45
6501 4F
6502 20
6503 47
6504 45
6505 4E
6506 49
6507 45
6508 20
6509 4C
650A 45
650B 56
650C 45
650D 4C
650E 20
650F 32
6510 20
6511 49
6512 4E
6513 54
6514 45
6515 52
6516 50
6517 52
6518 45
6519 54
651A 45
651B 52
651C 20
651D 45
651E 58
651F 54
6520 45
6521 4E
6522 53
6523 49
6524 4F
6525 4E
6526 20
6527 31
6528 0D       06000          DEFB   13
6529 42       06010          DEFM   'BY ANDREW HOWARD 25/02/83'
652A 59
652B 20
652C 41
652D 4E
652E 44
652F 52
6530 45
6531 57
6532 20
6533 48
6534 4F
6535 57
6536 41
6537 52
6538 44
6539 20
653A 32
653B 35
653C 2F
653D 30
653E 32
653F 2F
6540 38
6541 33
6542 0D00     06020          DEFW   13
6544 CDC901   06030 INIT     CALL   1C9H
6547 21F664   06040          LD     HL,TITLE
654A CD752B   06050          CALL   2B75H
654D 3A0340   06060          LD     A,(4003H)
6550 2A0440   06070          LD     HL,(4004H)
6553 322F60   06080          LD     (VECTOR),A
6556 223060   06090          LD     (VECTOR+1),HL
6559 3ABB41   06100          LD     A,(41BBH)
655C 2ABC41   06110          LD     HL,(41BCH)
655F 329762   06120          LD     (RUNEXT),A
6562 229862   06130          LD     (RUNEXT+1),HL
6565 3A5B41   06140          LD     A,(415BH)
6568 2A5C41   06150          LD     HL,(415CH)
656B 329963   06160          LD     (DEFEXT),A
656E 229A63   06170          LD     (DEFEXT+1),HL
6571 3EC3     06180          LD     A,195
6573 320340   06190          LD     (4003H),A
6576 32BB41   06200          LD     (41BBH),A
6579 325B41   06210          LD     (415BH),A
657C 2AA440   06220          LD     HL,(40A4H)
657F 2B       06230          DEC    HL
6580 E5       06240          PUSH   HL
6581 065E     06250          LD     B,94
6583 DD21B765 06260          LD     IX,DEST
6587 FD217366 06270          LD     IY,ADDR
658B E5       06280 RELOC    PUSH   HL
658C DD5E00   06290          LD     E,(IX+0)
658F DD5601   06300          LD     D,(IX+1)
6592 19       06310          ADD    HL,DE
6593 FD5E00   06320          LD     E,(IY+0)
6596 FD5601   06330          LD     D,(IY+1)
6599 EB       06340          EX     DE,HL
659A 73       06350          LD     (HL),E
659B 23       06360          INC    HL
659C 72       06370          LD     (HL),D
659D FD23     06380          INC    IY
659F FD23     06390          INC    IY
65A1 DD23     06400          INC    IX
65A3 DD23     06410          INC    IX
65A5 E1       06420          POP    HL
65A6 10E3     06430          DJNZ   RELOC
65A8 D1       06440          POP    DE
65A9 210060   06450          LD     HL,TEST
65AC 01F604   06460          LD     BC,TITLE-6000H
65AF EDB0     06470          LDIR
65B1 CD4D1B   06480          CALL   1B4DH
65B4 C37200   06490          JP     72H
65B7 0000     06500 DEST     DEFW   0
65B9 4F00     06510          DEFW   CMDTBL-6000H
65BB 2800     06520          DEFW   NXTCMD-6000H
65BD 3B00     06530          DEFW   FOUND-6000H
65BF 7A00     06540          DEFW   ADDRS-6000H
65C1 BA00     06550          DEFW   REPEAT-6000H
65C3 5D01     06560          DEFW   UNTIL-6000H
65C5 3F01     06570          DEFW   CANCEL-6000H
65C7 AD02     06580          DEFW   CALL-6000H
65C9 0603     06590          DEFW   DEFLAB-6000H
65CB 6F03     06600          DEFW   JUMP-6000H
65CD 9000     06610          DEFW   REPPNTM6000H
65CF BA00     06620          DEFW   REPEAT-6000H
65D1 C601     06630          DEFW   ERROR-6000H
65D3 9000     06640          DEFW   REPPNTM6000H
65D5 9000     06650          DEFW   REPPNTM6000H
65D7 9000     06660          DEFW   REPPNTM6000H
65D9 9200     06670          DEFW   REPTBL-6000H
65DB 9000     06680          DEFW   REPPNTM6000H
65DD 9000     06690          DEFW   REPPNTM6000H
65DF 9200     06700          DEFW   REPTBL-6000H
65E1 C601     06710          DEFW   ERROR-6000H
65E3 9000     06720          DEFW   REPPNTM6000H
65E5 C201     06730          DEFW   ERR-6000H
65E7 C301     06740          DEFW   ERREXT-6000H
65E9 C401     06750          DEFW   ERREXT-5FFFH
65EB E301     06760          DEFW   ERROR2-6000H
65ED C301     06770          DEFW   ERREXTM6000H
65EF C401     06780          DEFW   ERREXTM5FFFH
65F1 C201     06790          DEFW   ERR-6000H
65F3 0802     06800          DEFW   ERRTBL-6002H
65F5 1602     06810          DEFW   MANYRP-6000H
65F7 2702     06820          DEFW   NOREP-6000H
65F9 3C02     06830          DEFW   NOLAB-6000H
65FB 8B02     06840          DEFW   NEWRUN-6000H
65FD 9200     06850          DEFW   REPTBL-6000H
65FF 9000     06860          DEFW   REPPNT-6000H
6601 F604     06870          DEFW   TITLE-6000H
6603 F804     06880          DEFW   TITLE-5FFEH
6605 C601     06890          DEFW   ERROR-6000H
6607 CA03     06900          DEFW   PROC-6000H
6609 9003     06910          DEFW   DEF-6000H
660B 0F03     06920          DEFW   FNDCMD-6000H
660D 0F03     06930          DEFW   FNDCMD-6000H
660F 7F03     06940          DEFW   TSTPRC-6000H
6611 7F03     06950          DEFW   TSTPRC-6000H
6613 A003     06960          DEFW   PRCPNTM6000H
6615 CA03     06970          DEFW   PROC-6000H
6617 C601     06980          DEFW   ERROR-6000H
6619 4C02     06990          DEFW   MANYPC-6000H
661B 1003     07000          DEFW   FINDLB-6000H
661D A003     07010          DEFW   PRCPNT-6000H
661F A003     07020          DEFW   PRCPNT-6000H
6621 A203     07030          DEFW   PRCTBL-6000H
6623 A003     07040          DEFW   PRCPNT-6000H
6625 0603     07050          DEFW   DEFLAB-6000H
6627 A003     07060          DEFW   PRCPNTM6000H
6629 A203     07070          DEFW   PRCTBL-6000H
662B A003     07080          DEFW   PRCPNT-6000H
662D 0904     07090          DEFW   END-6000H
662F 7F03     07100          DEFW   TSTPRC-6000H
6631 C601     07110          DEFW   ERROR-6000H
6633 6002     07120          DEFW   NOPRC-6000H
6635 A003     07130          DEFW   PRCPNTM6000H
6637 A203     07140          DEFW   PRCTBL-6000H
6639 A003     07150          DEFW   PRCPNT-6000H
663B 2604     07160          DEFW   MSWHIL-6000H
663D 7B03     07170          DEFW   MSPROC-6000H
663F 8403     07180          DEFW   TSTPRC-5FFBH
6641 1003     07190          DEFW   FINDLB-6000H
6643 1E01     07200          DEFW   TEMP-6000H
6645 1E01     07210          DEFW   TEMP-6000H
6647 8504     07220          DEFW   WHILE-6000H
6649 CF02     07230          DEFW   EXEC-6000H
664B 7502     07240          DEFW   NOENDW-6000H
664D 2604     07250          DEFW   MSWHIL-6000H
664F 8403     07260          DEFW   TSTPRC-5FFBH
6651 CC02     07270          DEFW   EXECFG-6000H
6653 CD02     07280          DEFW   EXECSTM6000H
6655 CC02     07290          DEFW   EXECFG-6000H
6657 CC02     07300          DEFW   EXECFG-6000H
6659 CD02     07310          DEFW   EXECST-6000H
665B A003     07320          DEFW   PRCPNTM6000H
665D CC02     07330          DEFW   EXECFG-6000H
665F 5801     07340          DEFW   CANC3-6000H
6661 9A01     07350          DEFW   LINPUTM6000H
6663 EC00     07360          DEFW   CANC0-6000H
6665 0603     07370          DEFW   DEFLAB-6000H
6667 6C04     07380          DEFW   WHLNO-6000H
6669 6C04     07390          DEFW   WHLNO-6000H
666B 6D04     07400          DEFW   TSTWNO-6000H
666D 6D04     07410          DEFW   TSTWNO-6000H
666F 6C04     07420          DEFW   WHLNO-6000H
6671 6C04     07430          DEFW   WHLNO-6000H
6673 0440     07440 ADDR     DEFW   4004H
```

## HOW THE PROGRAM WORKS

Most of the program is self-explanatory. I shall therefore restrict my description to the more obscure points.

**Initialisation program (INIT), lines 6030 to 6490.** This program firstly sets all the vectors for the extension routines. Lines 6220 to 6490 then relocate the main program to below BASIC but above anything else. BASIC is moved up by resetting the pointer to the start of the BASIC program area (40A4H) and calling the NEW routine (at 1B4DH). Control is then returned to the READY message (JP 72H). The large data tables following are used in changing all the absolute addresses in the main program, according to the program's new address location.

**Test for VBBAS1 command routine (TEST) lines 20 to 560.** This routine is an extension to the RST 10H routine. The Execution Driver (ED) program in ROM which executes a line of BASIC performs RST 10H. The ED is located at 1D5AH. Upon execution of RST 10H control is transferred to TEST. Lines 20 to 90 make sure that execution mode is progress. If not, then control is transferred to the real RST 10H routine at 1D78H.

Through the use of tables lines 100 to 410 check that the next item to be executed is a VGBAS1 command. If not, then control is returned to the ED via 1D78H.

**REPEAT routine (REPEAT), lines 910 to 1110.** This routine is straightforward. It uses an internal buffer (REPTBL) of 40 bytes to hold the current address and the line number of the loop start. On entry to all VGBAS1 routines, HL contains the current address (ie the next byte address, in the BASIC program). REPPNT is an index into this table.

**CANCEL routines (CANCEL, CANC0, CANC1, CANC2 and CANC3), lines 1120 to 1700.** The CANCEL routine calls the ROM subroutine at 2B02H. This evaluates the expression, converts the result to integer and puts it in the DE register pair (D=high byte, E=low byte). Lines 1560 to 1640 then execute the appropriate CANC routine.

It should be noted that the 'R' labels, for example R61 in line 1430, are used by the INIT program for relocating the main program.

**UNTIL routine (UNTIL), lines 1710 to 1960.** Lines 1710 to 1780 check that there is a REPEAT-UNTIL loop in progress — error if not. The call to 2337H evaluates the expression and the call to 0994H tests for TRUE/FALSE condition. Since true=1, or non-zero, and false=0, zero, then line 1840 takes care of the loop.

**LINPUT routine (LINPUT), lines 1970 to 2170.** The first three lines set return address for the LET routine. The call to 2828H checks for illegal direct while 21CDH prints the message if one is supplied. The routine at 260DH gets the address of the variable into DE and 0AF4H checks that the variable is a string variable (TM error if not). Lines 2050 and 2100 then receive the input — 361H receives a line of input from the keyboard, and sets the carry (C) flag if Break was pressed. Line 2100 calls the STOP/END routine if this is the case. Lines 2110 to 2170 then register the input — 2868H builds a literal string pool entry and 1F32 is the LET routine.

**ERROR routine (ERROR), lines 2180 to 2660.** On entry to this routine, the A register contains the error number (1 to 6). The routine then executes the ROM error routine by performing a syntax error. However, the ?SN error is not actually printed because lines 2220 to 2280 set a vector, so that just before the error is printed, control is returned to the VGBAS1 error routine, ERROR2. Lines 2310 to 2340 reset the vector.

**Table reset routine (NEWRUN), lines 2670 to 2830.** Lines 2670 to 2740 make sure that the NEW/RUN command is in execution. If not, then control is returned to the calling routine or to Level III, whichever is appropriate, via the RUNEXT vector, set by the INIT routine.

Lines 2760 to 2820 reset the internal REPEAT-UNTIL/PROC buffers and cancel EXEC mode. Control is then returned to the caller/Level III.

**CALL routine (CALL and CALL2), lines 2840 to 3010.** Lines 2840 to 2870 check for absolute addressing mode. The absolute routine, lines 2880 to 2910 converts the address to binary and places the result in DE (this is done by the call to 1E5AH) before executing the subroutine entry routine, CALL 2.

The expression evaluation routine calls the ROM expression evaluator 2337H, and then converts the result into binary integer (call 0A7FH). The final address is then placed in DE for the CALL2 routine.

CALL2 transfers control to the machine code subroutine, whose address is in DE.

**EXEC routine (EXEC), lines 3040 to 3360.** Lines 3040 to 3090 evaluate the expression, test for parentheses, set EXEC mode and store the current address into the BASIC program. The subroutine at 29D7H tests the result for string (TM error if not) and on exit, HL contains the address of a three byte 'string header' containing the string's length (first byte) and a pointer to the start of the string (bytes 2 and 3). Lines 3110 to 3190 set BC to the length of the string and DE to the start of the string (its address). If no string has been specified (EXEC("")) then a syntax error results.

**! routine (DEFLAB), lines 3370 3390.** This routine skips to the end of the label by finding the next colon or end of line (the call to 1F0BH), and then executes the BASIC program from there.

**JUMP routine (JUMP), lines 4120 to 4150.** This routine finds the '!' command in the program with a matching label, via the FINDLB routine (used by JUMP and PROC).

**Find label routine (FINDLB), lines 3140 to 4110.** On entry to this routine the D register must contain the command to be searched for (this will be 21H for '!' or 0B0H for DEF) and HL must point to the label following the JUMP or PROC. Lines 3140 to 3470 set BC to point to the label to be found, HL to the start of the BASIC program, and the variable FNDCMD to the command to be found.

BASIC programs are stored in a 'queue' fashion, for example:

```
10 CLS
20 PRINT "HELLO"
```

These two lines would be stored as:

| | |
|---|---|
| 42E9 EF | — these two bytes are pointer to next line |
| 42EA 42 | |
| 42EB 0A | — these two bytes are line number |
| 42EC 00 | |
| 42ED 84 | — this byte is the CLS command |
| 42EE 00 | — this byte is the end of line marker |
| 42EF FC | — this is the next line. These two bytes point to the following line. |
| 42F0 42 | |
| 42F1 14 | — this is the line number (0014 is hex for 20) |
| 42F2 00 | |
| 42F3 B2 | — this is the PRINT command |
| 42F4 22 | — this is a quote |
| 42F5 48 | — 'H' |
| 42F6 45 | — 'E' |
| 42F7 4C | — 'L' |
| 42F8 4C | — 'L' |
| 42F9 4F | — 'O' |
| 42FA 22 | — end quote |
| 42FB 00 | — end of line marker. |
| 42FC 00 | — this would be the pointer to the next line, |
| 42FD 00 | — but as it is 0000 it signifies the end of the program. |

After line 3470, HL points to the start of the program, and in this exmple, HL would contain 42E9H. However, it should be noted that with VGBAS1 present BASIC programs start higher up in memory.

Lines 3480 to 3510 make sure that the end of the program has not been reached, and if so, an error is printed ("UNDEFINED LABEL").

Lines 3520 to 3590 set DE to the line number for that line, HL to the start of the text in that line (in the above example, that would be 42EDH or 42F3H), and B to the first character in the line. Line 3600 compares this character with the command to be looked for, and if the same, executes the compare label routine, LABEL. Otherwise, lines 3640 to 3690 set HL to point to the line and the program loops round to test this next line.

Lines 3720 to 3800 are executed when the correct command has been found, and set HL to point to the label after the command (skipping over the word PROC if it is there).

```
6675 ØD6Ø        Ø745Ø        DEFW    RØ+1
6677 236Ø        Ø746Ø        DEFW    R1+1
6679 356Ø        Ø747Ø        DEFW    R2+1
667B 436Ø        Ø748Ø        DEFW    R3+1
667D 7A6Ø        Ø749Ø        DEFW    ADDRS
667F 7C6Ø        Ø75ØØ        DEFW    ADDRS+2
6681 7E6Ø        Ø751Ø        DEFW    ADDRS+4
6683 8Ø6Ø        Ø752Ø        DEFW    ADDRS+6
6685 826Ø        Ø753Ø        DEFW    ADDRS+8
6687 846Ø        Ø754Ø        DEFW    ADDRS+1Ø
6689 BC6Ø        Ø755Ø        DEFW    R4+1
668B BF6Ø        Ø756Ø        DEFW    R5+1
668D C86Ø        Ø757Ø        DEFW    R6+1
668F CE6Ø        Ø758Ø        DEFW    R7+2
6691 E66Ø        Ø759Ø        DEFW    R8+2
6693 EE6Ø        Ø76ØØ        DEFW    R9+1
6695 F16Ø        Ø761Ø        DEFW    R1Ø+1
6697 FF6Ø        Ø762Ø        DEFW    R11+1
6699 5F61        Ø763Ø        DEFW    R12+1
669B 6261        Ø764Ø        DEFW    R13+1
669D 6B61        Ø765Ø        DEFW    R14+1
669F 7B61        Ø766Ø        DEFW    R15+2
66A1 C761        Ø767Ø        DEFW    ERROR+1
66A3 DØ61        Ø768Ø        DEFW    R16+1
66A5 D361        Ø769Ø        DEFW    R17+1
66A7 D861        Ø77ØØ        DEFW    R18+1
66A9 E561        Ø771Ø        DEFW    R19+1
66AB E861        Ø772Ø        DEFW    R2Ø+1
66AD F161        Ø773Ø        DEFW    R21+1
66AF F861        Ø774Ø        DEFW    R22+1
66B1 ØA62        Ø775Ø        DEFW    ERRTBL
66B3 ØC62        Ø776Ø        DEFW    ERRTBL+2
66B5 ØE62        Ø777Ø        DEFW    ERRTBL+4
66B7 BC41        Ø778Ø        DEFW    41BCH
66B9 9C62        Ø779Ø        DEFW    R23+1
66BB 9F62        Ø78ØØ        DEFW    R24+1
66BD A44Ø        Ø781Ø        DEFW    4ØA4H
66BF F94Ø        Ø782Ø        DEFW    4ØF9H
66C1 3963        Ø783Ø        DEFW    R25+1
66C3 866Ø        Ø784Ø        DEFW    ADDRS+12
66C5 5C41        Ø785Ø        DEFW    415CH
66C7 1963        Ø786Ø        DEFW    R31+1
66C9 2963        Ø787Ø        DEFW    R32+1
66CB 9263        Ø788Ø        DEFW    R33+1
66CD 3F63        Ø789Ø        DEFW    R34+1
66CF D263        Ø79ØØ        DEFW    R36+1
66D1 D563        Ø791Ø        DEFW    R37+1
66D3 DF63        Ø792Ø        DEFW    R38+1
66D5 1Ø62        Ø793Ø        DEFW    ERRTBL+6
66D7 7263        Ø794Ø        DEFW    R39+1
66D9 E463        Ø795Ø        DEFW    R4Ø+2
66DB FF63        Ø796Ø        DEFW    R41+2
66DD A262        Ø797Ø        DEFW    R42+1
66DF A562        Ø798Ø        DEFW    R43+1
66E1 9E63        Ø799Ø        DEFW    R44+1
66E3 Ø761        Ø8ØØØ        DEFW    R45+1
66E5 ØA61        Ø8Ø1Ø        DEFW    R46+1
66E7 1861        Ø8Ø2Ø        DEFW    R47+1
66E9 886Ø        Ø8Ø3Ø        DEFW    ADDRS+14
66EB 2D64        Ø8Ø4Ø        DEFW    R48+1
66ED 3F64        Ø8Ø5Ø        DEFW    R5Ø+1
66EF 1262        Ø8Ø6Ø        DEFW    ERRTBL+8
66F1 3364        Ø8Ø7Ø        DEFW    R51+1
66F3 3664        Ø8Ø8Ø        DEFW    R52+1
66F5 4364        Ø8Ø9Ø        DEFW    R53+1
66F7 5F64        Ø81ØØ        DEFW    R54+1
66F9 8Ø63        Ø811Ø        DEFW    TSTPRC+1
66FB 6464        Ø812Ø        DEFW    R55+1
66FD CD63        Ø813Ø        DEFW    R56+1
66FF 3A61        Ø814Ø        DEFW    R6Ø+1
67Ø1 2461        Ø815Ø        DEFW    R61+1
67Ø3 8A6Ø        Ø816Ø        DEFW    ADDRS+16
67Ø5 8C6Ø        Ø817Ø        DEFW    ADDRS+18
67Ø7 1462        Ø818Ø        DEFW    ERRTBL+1Ø
67Ø9 B564        Ø819Ø        DEFW    SKPEND+1
67ØB BA64        Ø82ØØ        DEFW    R62+1
67ØD D762        Ø821Ø        DEFW    R63+1
67ØF DA62        Ø822Ø        DEFW    R64+1
6711 ØE64        Ø823Ø        DEFW    R65+1
6713 1564        Ø824Ø        DEFW    R66+1
6715 1964        Ø825Ø        DEFW    R67+1
6717 4A64        Ø826Ø        DEFW    R68+1
6719 5A61        Ø827Ø        DEFW    R69+1
671B A862        Ø828Ø        DEFW    R7Ø+1
671D 8E6Ø        Ø829Ø        DEFW    ADDRS+2Ø
671F 7761        Ø83ØØ        DEFW    R71+1
6721 6A64        Ø831Ø        DEFW    R72+1
6723 8F64        Ø832Ø        DEFW    R73+1
6725 8764        Ø833Ø        DEFW    R74+1
6727 8A64        Ø834Ø        DEFW    R75+1
6729 C564        Ø835Ø        DEFW    R76+1
672B BF64        Ø836Ø        DEFW    R77+1
672D CA64        Ø837Ø        DEFW    R78+1
6544             Ø838Ø        END     INIT
ØØØØØ TOTAL ERRORS
```

```
ADDR     6673
ADDRS    6Ø7A
CALL     62AD
CALL2    62C5
CAN2     6ØFA
CAN3     6113
CANCØ    6ØEC
CANC1    61Ø5
CANC2    612Ø
CANC3    6158
CANCEL   613F
CHRMTC   6Ø32
CMDLP    6Ø13
CMDLP1   6Ø16
CMDTBL   6Ø4F
CPLABL   634C
DEF      639Ø
DEFEXT   6399
DEFLAB   63Ø6
DEFPRC   639C
DEST     65B7
END      64Ø9
ENDCMD   642B
ENDNRM   641E
ENDPRC   6441
ENDWHL   645D
ERR      61C2
ERREXT   61C3
ERROR    61C6
ERROR2   61E3
ERRTBL   62ØA
EXEC     62CF
EXECFG   62CC
EXECST   62CD
EXP      62BC
FINDLB   631Ø
FNDCMD   63ØF
FOUND    6Ø3B
INIT     6544
JPLOOP   631B
JUMP     636F
LABEL    633B
LABEL2   6348
LABMTC   6354
LBMTC1   635E
LBMTC2   6366
LINPUT   619A
MANYPC   624C
MANYRP   6216
MSPROC   637B
MSWHIL   6426
NEWRUN   628B
NOENDW   6275
NOLAB    623C
NOLABL   6336
NOLBMT   632F
NOPRC    626Ø
NOREP    6227
NOTEXC   6ØØ9
NOTRUN   6294
NOWHIL   64ED
NOWLNO   64CF
NXTCMD   6Ø28
PRCDEF   6347
PRCPNT   63AØ
PRCTBL   63A2
PROC     63CA
PROC2    63E1
RØ       6ØØC
R1       6Ø22
R1Ø      6ØFØ
R11      6ØFE
R12      615E
R13      6161
R14      616A
R15      6179
R16      61CF
R17      61D2
R18      61D7
R19      61E4
R2       6Ø34
R2Ø      61E7
R21      61FØ
R22      61F7
R23      629B
R24      629E
R25      6338
R3       6Ø42
R31      6318
R32      6328
R33      6391
R34      633E
R36      63D1
R37      63D4
R38      63DE
R39      6371
R4       6ØBB
R4Ø      63E2
R41      63FD
R42      62A1
R43      62A4
R44      639D
R45      61Ø6
R46      61Ø9
R47      6117
R48      642C
R5       6ØBE
R5Ø      643E
R51      6432
R52      6435
R53      6442
R54      645E
R55      6463
R56      63CC
R6       6ØC7
R6Ø      6139
R61      6123
R62      64B9
R63      62D6
R64      62D9
R65      64ØD
R66      6414
R67      6418
R68      6449
R69      6159
R7       6ØCC
R7Ø      62A7
R71      6176
R72      6469
R73      648E
R74      6486
R75      6489
R76      64C4
R77      64BE
R78      64C9
R8       6ØE4
R9       6ØED
RELOC    658B
REP2     6ØCA
REPEAT   6ØBA
REPPNT   6Ø9Ø
REPTBL   6Ø92
RUN2     629A
RUNEXT   6297
SKIP     6Ø2Ø
SKPEND   64B4
SRCHWL   64A2
TEMP     611E
TEST     6ØØØ
TITLE    64F6
TSTLP    6384
TSTLPM   638A
TSTPRC   637F
TSTWNO   646D
UNT2     616D
UNTIL    615D
VECTOR   6Ø2F
WHILE    6485
WHILE2   6493
WHLNO    646C
```

**DEF routine (DEF), lines 4290 to 4370.** This routine tests for the presence of the word 'PROC' following the DEF. If it is there, then the '!' routine is executed, skipping over the label. Otherwise, control is either returned to BASIC producing a syntax/Level III error or to Level III.

**PROC routine (PROC), lines 4400 to 4660.** The first four lines of this routine find the address (in HL) and line number (in DE) of the appropriate DEFPROC statement. Like REPEAT-UNTIL, PROC uses an internal buffer of 40 bytes (to allow 10 levels) to store the present address and line number.

**END routines (END, ENDNRM, ENDPRC and ENDWHL), lines 4670 to 5170.** This routine handles all END functions. The first three lines check for any suffix (ie check for ENDPROC/ENDWHILE), and so transfer control to the end-command routine, ENDCMD. Otherwise, lines 4700 to 4720 check for EXEC mode. If this is in force, then EXEC mode is cancelled, and control is returned to the statement following the EXEC. If EXEC mode is not in force, then the return address is placed on the stack and the normal END routine in ROM is executed (1DAFH).

**WHILE routine (WHILE), lines 5360 to 5970.** This routine first sets the value for expression1 (or the WHLNO) to zero. Then, if there is an expression1 it is evaluated, converted to integer and tested to make sure that it is 255 or less (overflow if not), and that it is enclosed in parentheses and followed by a comma. The result is then stored as WHLNO.

The routine at 1F07H is the REM routine and after line 5490 HL points to the next line pointer, following the end of line marker of the line containing the WHILE. As in FINDLB, lines 5500 and 5550 check that "NO ENDWHILE STATEMENT" has occurred or not.

This concludes the article on VGBAS1, a program providing 13 powerful new commands. VGBAS1 could be the first in a series of interpreter extensions programs, each enhancing the BASIC language that I hardly ever use!

# HAVING A BRAINSTORM

**Henry Budgett**

For an industry which contains rather more than its fair share of pundits and 'experts' the microcomputer world seldom seems to relate itself to reality. We are always reading of new software developments and new languages; all the fuss about FORTH seems a little misplaced when you realise that it was originally developed around 1970 on an IBM 1130! The number of articles and features that have been written over the last six years concerning the development and usage of micros in this country alone makes an impressive pile, yet the majority of the 'discoveries' can be directly related to software that exists on minicomputers.

There have been few real breakthroughs in software for micros — the first word processing package was almost certainly a cut-down mini program — but one that stands out from the crowd is VisiCalc. This, so the story goes, was written primarily to show that it was possible to create financial models on a small computer — something that was thought to be totally impractical due to the limitations of processing speed and memory.

The piece of software that's the subject of this review is, in many ways, a completely new type of program. That is to say it has no directly obvious predecessor in either the mainframe or minicomputer market. However, as if to confuse the issue, the program does contain elements that are instantly recognisable as being derived from word processors and data bases. It is the engineering of the complete package that makes this product so remarkable and, for the moment at least, unique.

Both word processors and spreadsheets are, at the most fundamental of levels, tools that can increase productivity. The advantages of using a word processor as opposed to a typewriter are generally obvious; I, for one, simply couldn't generate the quantities of material I do without one (the quality is something beyond the machine's control!). A spreadsheet is really a word processor for someone who deals with figures and their relationships. Once a user has mastered, or come to terms with, a program of either type the increase in both productivity and efficiency can pay for the package in a very short period of time.

A similar tool for increasing efficiency is a data base. Here information is stored, either as an index card or as a single record. Items within each card or record can be accessed by the program and relationships established between seemingly unconnected pieces of information. A data base program simply takes over the management of large quantities of information from manual, paper-based systems and, if properly established, makes the retrieval of any item more efficient. Unfortunately there is a snag. To set up a data base to handle information in the way you require needs a great deal of forethought and planning. Altering the way records are held in the computer once the data has been keyed in is generally a very messy business. It is often worth investing a great deal of time in structuring the information before the data base is established.

## DATA STRUCTURE

It is this structuring of information that people find difficult. A set of figures produced by a spreadsheet may be totally useless because the structure is different to that required by the company auditor. Documents created on a word processor may be completely accurate yet totally lacking in structure. A data base may have to be scrapped because the information has been stored in such a way that the user cannot retrieve certain items. What would be rather useful is a program that allowed the structure to be developed first. BrainStorm is a program that sets out to achieve this by creating a data base that consists of the contents of your mind. No, it's got nothing to do with the film of the same name, but it does allow you to really get your ideas down on paper.

You might have expected such a new product to have been developed by one of the software giants but not only is it British, the company that markets it is just two years old. Founded by David Tebbutt (one-time Editor of **Personal Computer World**), Bill Barrow and Alan Wood with the intention of producing the best in British software, Caxton had just three packages in its catalogue. Each of these represents, if not the market leader, the top of the field.

Caxton's own subtitle for their new product is 'The Ideas Processor', a phrase introduced to provide a link with the already understood concept of word processing. As a generic title I suppose that it works as well as any but BrainStorm cannot really 'process' your ideas. What the program does is to allow you to enter all your thoughts on a subject into a computer. Once entered these apparently unstructured and random ideas can be manipulated and edited into a coherent and structured model of your thoughts.

The traditional method for structuring your ideas before planning a marketing campaign, writing a feature article, writing a report and so on, is to grab loads of paper and spend hours scribbling. Drawbacks with this method are legion. It is possible to completely miss out sections because you cannot see an overall picture at any one time, duplications and repetitions often appear and the only solution tends to be to use the first draft as the structure for the second and so on . . . Even with word processors providing the typing power the process tends to be very slow and extremely inefficient.

Using BrainStorm overcomes most of these problems without forcing the user to be structured in approach. Each set of ideas, or 'Mind Model' as I understand it, is given a title, in the case of the plan for writing this feature it was called 'BrainStorm Review'. Each model can only have one title but there can be as many subtitles as there is memory left. A constant check is kept on memory usage and the amount free is constantly displayed on the screen. If a model should get too large then it can be split at some appropriate point and stored on disc.

## GETTING PROMOTED

The entry of information into BrainStorm is line-based rather than the familiar free entry used on word processors. The reason is that every entry is (albeit conceptually) indexed to all other entries and it would be impossible to achieve this with continuous text. In creating a model it is often easiest to outline the whole structure so the first set of entries in the Review model look like Fig. 1. Each entry can now be made into the heading for a further set of entries. The method for achieving this is simple; position the cursor over the entry you wish to add more ideas to and 'Promote' it to a heading. This clears the screen and places the entry at the top of a fresh screen. All the entries typed in under a heading are called descendants and these, in turn, can be Promoted to being headings for further sets of entries.

## NAMESAKES

If BrainStorm spots an entry that is identical to one held elsewhere in the model it signals this by placing a number next to the new entry. The first duplication would have a '1', the second a '2' and so on. These duplications are called Namesakes and it is possible to jump directly from one to any other. If you find that an entry would make more sense in a different position it can be moved, complete with all its descendants, to the new position. This allows you to re-model your thoughts at any stage of the input or even days later.

```
BrainStorm Review
Introduction
Philosophy behind Caxton
Related Caxton products
What is an 'Ideas Processor'
How BrainStorm works
What to use an 'Ideas Processor' for
Summary of features
Conclusion
```

**Fig. 1 an inital set of entries.**

BrainStorm also allows entries to be edited and amended at any level: all the updating of the descendants is handled automatically where necessary.

It is also possible to Hunt for entries within a model. Wildcard facilities are provided, allowing single or multiple character sequence searches to be carried out. The facility can also be extended to create multiple entries by prefixing the first occurrence with a suitable character and then forcing the system to reproduce the character string at the current position. Because all the entries will be namesakes removing the initial character from one of them will remove it from them all.

Models created by BrainStorm can be merged to create a joint model. This facility was actually used during the three-year development of the program to allow various members of the Caxton team to create models of their thoughts on the product and then merge them to find the common elements. Text files can be read into a model as well, allowing standard documents, contracts and so on to be created from a single model. Text files are obviously not structured in the same way as BrainStorm's own files but they can be converted into BrainStorm models simply by using the manipulative commands, Put and Get.

## USE (AND ABUSE!)

It is almost impossible to outline all the possible uses for BrainStorm simply because new ones are always cropping up. Any creative process where a logical structure is needed can be tackled. The obvious examples are reports, documentation, features, articles, and so on. However, the list doesn't end here because sales and advertising campaigns, product launches and even manufacturing processes can all be developed as BrainStorm models, often making use of the merge facility to allow several people the option to add in their own thoughts.

On the more esoteric level it is possible to design the structure for a data base on BrainStorm, hopefully eliminating the problems connected with data bases designed by the programmer rather than the user. BrainStorm would allow the user to create the original specification first, once all the structures had been thoroughly checked. It is also possible to write programs with BrainStorm: indeed, the manual comes with just such an example. Once written the files can simply be passed across as text output and then RUN.


**Fig. 2 An annotated guide to promotions.**

Although BrainStorm is currently only available under CP/M it will run on virtually any CP/M system known. Regular readers of **Computing Today** may raise their eyebrows on hearing that — my dislike of CP/M in general is well known. However, it is probably a measure of the regard I have for Caxton products that I only bought a CP/M card for my Apple in order to use both BrainStorm and Cardbox. No other CP/M programs have yet been run and it is probable that they never will! My main objections to CP/M are based on the general unreliability of its operation: the famous BDOS Error on a non-existent disc drive is a prime example that has lost me more Wordstar files than I care to think about. Caxton have, in a rather subtle way, exploited one of the many bugs within CP/M to provide a rescue route for anyone caught by such a disaster. In the event of a crash through CP/M simply type REBRAIN and, in most cases, your model will be recovered for you. Now why can't other software houses do the same?

## CONFIGURATIONS

The configuration facility supplied with the program allows any one of a number of standard terminals to be used, or you can format the system to suit your own particular needs. The copy of BrainStorm supplied for this review came configured for an Apple II system running the Microsoft SoftCard CP/M system. In order to get an 80-column display the Videx card was being used, a perfectly normal set-up. Unfortunately, my Apple IIe uses Apple's own 80-column card and the display format for graphics is slightly different. Using the configurator it proved to be a matter of a couple of minutes work to change the program to work with my Apple. Encouraged by this I decided to alter all the editing functions from those designed for the upper-case-only Apple II to the full keyboard of the IIe. Once again there was no problem and my version of BrainStorm bears little resemblance in terms of command keys to the original. Needless to say, all the changes were made to a backup copy and not to the original . . .

The manual supplied with BrainStorm is, with one exception, excellent. Unlike previous Caxton products the manual is ring-bound and housed in an outer box with the disc. Broken into three main sections, the manual starts with a quick tutorial session to get you going. The main reference section is next with more detailed information on the commands and the final part covers the configuration program. The final pages of the binder contain a number of worked examples, the basis for an interactive diary system is also supplied on the disc, and there is a quick reference chart.

The only section of the manual which doesn't really fulfil its purpose is the explanation of the print formats. Now, it may be that I'm even more retarded than I thought but I still cannot fathom out the various permutations. A long chat with the co-author and originator, David Tebbutt, still hasn't really cleared the mystery. As there are some 2.5 million possible permutations I feel that either more built-in options should be provided or the number of variations reduced to make things a little easier.

## BRAINSTORMING

In a review such as this it is often very difficult to get across the nature of the beast. Because I use BrainStorm to organise my thoughts before I turn to the word processor and turn them into text it is possible that this has influenced the way in which the review has been written. One of the criticisms raised against BrainStorm at its launch was that it would reduce creativity in writing. While this point of view is understandable it is definitely invalid. I do all the creative writing at the keyboard of the word processor, not with BrainStorm.

Even more difficult to get across is the enormous range of organisational tasks that BrainStorm can tackle. Almost any job which requires the use of a structured approach could benefit from using the program. Ideas about what it could be used for tend to occur during conversations on completely unrelated topics. During a discussion on medical interviewing techniques it suddenly became obvious that the method we were investigating was eminently suitable for use in conjunction with BrainStorm. Caxton are so keen to find out what people are using it for that one in every hundred purchasers who register will get their money back, incentive indeed at £295.

## CONCLUSIONS

Britain has always enjoyed an excellent reputation for its innovative software; our transatlantic cousins build the hardware but we program it. With BrainStorm it would be very nice to think that another British product had joined the front runners in the market place. Unfortunately, because of its innovation it is going to be hard work for Caxton to get the concept of Ideas Processors across to a market which is only just becoming at ease with word processors and regards the spreadsheet as something really clever. If the idea **can** be got across successfully — and the amount of imitation will be a definite measure of this — then we could well be on the verge of a new era in software.

# DISASSEMBLY TECHNIQUES

Don Thomasson

Whenever I produce a new listing or analysis of a piece of software, which happens fairly often, someone always wants to know how it is done. In many cases they have tried to disassemble a program, and have run into difficulty. They want to know where they went wrong. Here then, is an outline of the procedures which I have found useful. As a basis for examples, I have used the BBC Operating System issue 1.2, since that presents some special problems that are worth examining.

Any workable computer program must preserve continuity. There will be a defined starting point, and perhaps more than one. From that there must be a clearly-defined path to every part of the program. If there is no traceable route to a particular area, that area must either contain data or 'dead code'.

The paths are made up of modules which begin at an identified starting point and continue until an absolute jump or return instruction is reached. Any jumps within the module will identify entry points to other modules, and so will subroutine calls. By tracing out each module inturn, it is possible to identify all the valid entry points, and hence, all the valid modules.

This can be a tedious task, and it is understandable that some explorers prefer to disassemble the whole program in bulk, and that can lead to confusion, especially where code follows a data area.

## TOOLS

To work out the valid modules effectively, some tools are needed. A disassembly routine is almost essential, unless you can remember all the instruction codes and translate them by hand, which needs practice and a lot of patience. It would help if the routine was also able to produce hexadecimal and alphabetical dumps of data areas, and for serious work an optional printing facility is a must.

As the work proceeds, you will accumulate a growing list of entry-point addresses. These will emerge in random order, and it is very useful to have a program which will sort them into the correct sequence. Avoid using the popular 'bubble sort', which is too slow for practical purposes. In many cases, the best system is to insert each new entry into its correct place in sequence, but that can become too slow with long lists. If other methods fail in this respect, a merging sort offers maximum speed. This forms the data into sequenced blocks. After the first pass there are two entries in a block. After the second pass there are four, and so on.

## THE 6502

It is important to be aware of any peculiarities of the processor involved, and the 6502 has several worth noting. First, there are no conditional absolute jumps, and no unconditional relative jumps. Consequently, it is common practice to simulate absolute conditional jumps by a conditional jump to an absolute jump. More confusing, nominally conditional relative jumps may be made absolute by pre-setting the stated condition. It is therefore uncertain, on occasion, whether a conditional jump is really conditional, or whether it ends a module.

Second, the 6502 programmer makes extensive use of the 'page 0' locations as quick-reference equivalents of on-board registers. Until the meaning of each location is discovered, this can lead to further uncertainty.

## STARTING POINTS

All that remains is to discover where to make a start, and in the case of the 6502 this involves examination of the last six bytes at the top of store. In the case of the BBC machine we find:

```
FFFA/B   &0D00   Non-maskable interrupt entry
FFFC/D   &D9CD   Initial start entry
FFFE/F   &DC1C   Interrupt entry, and BRK entry
```

Here we have three possible start points. It will be found that the first is of limited interest, as an RTI instruction is set at &0D00. There is a strong case for beginning with the second entry, since this will show how the variables are initialised, but to be perverse, a start will be made with the third entry, for a reason which will soon emerge.

This entry is used in response to either a maskable interrupt or a BRK instruction. In the latter case, the B flag is set, and this must be checked by the first module, which is found to be:

```
DC1C    STA &FC
DC1E    PLA
DC1F    PHA
DC20    AND #10
DC22    BNE &DC27
DC24    JMP (&0204)
```

From this, we can identify a direct entry point at &DC27, and an indirect link via &0204.

Both the BRK instruction and a maskable interrupt put a return address on the stack, then the contents of the flag register. The latter can be popped into A, the previous contents of A having been saved in &00FC. The stack is restored by pushing A, and bit 4 of the saved flag register byte is then checked. If the bit is true, a BRK instruction is involved, and the routine jumps to &DC27. Otherwise it indirects via (&0204). A check on the contents of that location and its neighbour show that the jump is normally to &DC93, but we must remember that this could be changed.

## CONTINUATION

We will concentrate on the definite entry at &DC27, the one which serves BRK. This leads to a 45-byte module which contains two subroutine calls and an indirect jump. This may be paraphrased.:

```
DC27    PUSH X
DC29    X =  S
DC2A    (&00FD) = (&0103 + X) - 2
DC33    (&00FE) = (&010 + X) - carry
DC3A    (&024A) = (&00F4)
DC3F    (&00F0) = X
DC41    X = 6
DC43    CALL &F168
DC46    X = (&028C)
DC49    CALL &DC16
DC4C    PULL X
DC4E    A = (&00FC)
DC50    CLEAR I
DC51    JMP (&0202)
```

To interpret this, we need to know some of the characteristics of the 6502. X is pushed. No problem. Then X = S, the stack pointer, and X is used as a displacement relative to &0103/4 to read two bytes forming a word, from which two is subtracted, the result being put in (&00FD/E).

Now, the normal stack base is &0100. By using &0103/4, we will access the third and fourth stack entries, and these will hold the return address set by BRK, which is the program counter value plus two. We therefore set the program counter value in (&00FD/E).

Until we have fathomed out the action of the two subroutines, we can say little more about the action of this module. X is saved in (&00F0), and then used to hold parameters for the subroutines. Then X is restored from the stack, A is restored from (&00FC); interrupt is cleared, ie enabled; and an indirect jump via (&0202) follows.

So far, what we have found means little. We need to explore further before we can discover what these actions mean. Since this is a side routine, rather than part of the main program flow, we may as well do that immediately; we find that (&0202) = &DC54, and:

```
DC54    Y = Ø
DC56    CALL DEB1
DC59    IF bit Ø of (&Ø267) = 1 then dynamic halt
DC5F    CALL FFE7
DC62    CALL FFE7
DC65    JMP &DBB8
```

DEB1 proves to be a string output routine, while FFE7 is Newline. The dynamic halt is a jump to the same location, from which the exit must be by an interrupt. DBB8 is in the initialisation routine.

We have therefore discovered that a BRK instruction causes a subsequent string to be output, and this is useful information, as it explains the otherwise mysterious intrusion of text data in the middle of code.

## INTERPRETATION

Examination of this section of code has been carried through at once, where it would normally be better to complete the link list first, the object being to show that the most complete knowledge of the actual code is by no means the end of the story. Sometimes the meaning of the code is only discovered through sheer inspiration, based on the look of the instructions. In other cases, the only useful route is through a careful tabulation of the available data.

In the case of the interrupt routine accessed by JMP(&0204), useful indications are obtained by noting the references to the peripheral devices, mainly checks on the interrupt registers to identify the source of the interrupt. Once the source has been identified, the appropriate action may be easy to guess.

It is for this reason that the BRK/interrupt vector was taken as the starting point. The initialising routine provides a lot of useful data, but the BRK and Interrupt functions can be even more illuminating.

A final point on interpretation is that the BBC Manual contains much useful data identifying certain specialised entry points, and it is worth matching these up with the entry points discovered by search. The identification of FFE7 as Newline came from that source . . .

## AUTOMATION

It has been suggested that the process of identifying entry points and hence defining valid code modules could be automated, but there are severe difficulties. Not the least arises when entry is by reference to a link table. The addresses in the table will not be picked up by the normal search process, and must be entered in the link list manually. In the case of the 6502, the uncertainty regarding nominally conditional jumps is a further complication, and the BBC machine adds a further difficulty by its specialised use of BRK.

A brave attempt to create a fully automated disassembler for a rather simpler type of machine has been examined, and it almost worked, but only for relatively short and simple programs.

## CONCLUSION

Summing up the process: first find the starting points, then trace all the other entries which should be sorted into sequence. Then, and only then, is it safe to begin disassembly proper, taking one module at a time.

During this process, ti may be necessary to look out for the use of link tables defining further entries, and any such entries must be added to the list.

When disassembly is complete, interpretation can start, and this is the hardest part of the whole task. In a way, it can be compared with the playing of an adventure game, but despite the ingenuity of some adventure writers their problems may be much easier to solve. . .

# SOME EXPERIMENTS WITH RECURSION

Dr. Barry Landsberg

**Recursion: (an article about various types of Recursion: (an article about various types of Recursion: (an article about various types of Recursion: (an . . .))))**

It is an old joke that if you look up the definition of recursion in a dictionary, it says **Recursion: see recursion.** In the world of computer programming, recursion is the act of a routine (which means subroutine, procedure or function depending on which language you program in) calling itself. This may happen directly, or indirectly. Direct recursion is when a subroutine calls itself from within itself, and indirect recursion is when a routine calls other routines which call the first routine again. Indirect recursion may be understood using another old joke about a piece of paper where PTO (please turn over) is written on both sides.

Before proceeding with examples of recursion it is best to describe briefly what happens when a routine is called. A subroutine call initiates a jump to another part of the program, but 'remembers' where the jump was made from. To 'remember' what is necessary a special area of memory must be reserved, and this is normally achieved using a 'stack' (which is an area of memory onto which you 'push' data when you want to store it, and 'pull' data when you want to retrieve it). When the called subroutine has reached its return (or end) command, it simply pulls off the stack the information it stored and the program continues from where it was before the procedure was called.

Now, consider what must happen when routine A calls routine B which calls routine C. When routine C is called, the stack must contain not only information about where in B to return to when C has finished (and also any local variables in B, but also where in A to return to when B has finished. In normal recursion, if A calls itself a large number of times, each call will push more information onto the stack. Bearing in mind that computers have only a limited amount of memory, this means that subroutine calling and recursion may only be made to a limited depth.

It is the purpose of this article to investigate how recursion is treated in the languages BASIC, Pascal, LISP and Logo as implemented on the Apple II microcomputer, and to show that Logo is the only language that allows, in certain cases, a routine to go on calling itself for ever! However, it might be instructive to consider how the Apple II handles subroutine calls and recursion in its native machine code.

## MACHINE CODE

The Apple is based on the 6502 microprocessor, and the easiest way to manipulate and test very short machine code routines is via a program resident in ROM called the monitor. To enter the monitor, it is only necessary to type CALL −151. and you will be greeted with a star (★) as your prompt. Now, we type in the crudest possible of recursive routines starting from (say) location $300 (the $ sign refers to a hexadecimal number) using the following 6502 code:

```
300:  20  00  03
```

which is the program JSR $300. When we execute it (by typing 300G) the Apple appears to 'hang' until we type <RESET> and re-enter the monitor in order to examine the stack.

The 6502 processor reserves the area of memory between $100 and $1FF for its stack, and we can see (by typing 100.1FF) that the stack mainly consists of the pattern 02 03 repeated. This is because the last executed code was $0302, and after such a subroutine call is completed, the program would know to continue executing from location $303. Thus such a subroutine call results in two bytes being pushed onto the stack.

Now we ask why the above instruction caused the Apple to hang. If the stack may hold only 256 bytes, surely after 128 subroutine calls something should go wrong? The following program prints out FF on the first time round, FE on the second and so on, in order for us to count the number of calls made before stack overflow:

```
300 - A2 00     LDX #$00
302 - CA        DEX
303 - 8A        TXA
304 - 20 DA FD  JSR $FDDA ; Prints accumulator in hex
307 - 20 8E FD  JSR $FD8E ; Print a line feed
30A - 20 02 03  JSR $0302 ; Recursive call
30D - 60        RTS       ; This is never reached!
```

When this program is run, it will carry on working for as long as you care to wait! Evidently the 6502 recycles its stack, and when all 256 bytes are used up it simply starts again and overwrites what was there before. Assembler programmers should beware of nesting subroutine calls too deeply as no error message seems to be given when the stack overflows, and unexpected things may occur when this does happen! Most high-level languages which do use this stack trap this error as will be sen when we experiment with BASIC.

```
                    PROGRAM 1a
                    ----------

        BASIC PROGRAM TO TEST DIRECT RECURSION

10 INPUT X
20 PRINT X
30 X=X-1
40 IF (X)0) THEN GOSUB 20
50 RETURN
60 END
                    PROGRAM 1b
                    ----------

        BASIC PROGRAM TO TEST INDIRECT RECURSION

10  INPUT X
20  PRINT X
30  X=X-1
40  IF (X)0) THEN GOSUB 120
50  RETURN
60  END
120 PRINT X
130 X=X-1
140 IF (X)0) THEN GOSUB 20
150 RETURN
160 END
```

```
                                PROGRAM 2a
                                ----------

              PASCAL PROGRAM TO TEST DIRECT RECURSION


PROGRAM DIRECTRECURSION:
VAR X:INTEGER:
         PROCEDURE RECURSE (X:INTEGER)
         BEGIN
              WRITELN(X):
              X:=X-1:
              IF (X)0) THEN RECURSE (X):
         END:
BEGIN
         READLN(X):
         RECURSE(X);
         WRITELN('FINISHED');
END.

                                PROGRAM 2b
                                ----------

              PASCAL PROGRAM TO TEST INDIRECT RECURSION


PROGRAM INDIRECTRECURSION:
VAR X:INTEGER:
PROCEDURE RECURS2(X:INTEGER);FORWARD:

     PROCEDURE RECURS1(X:INTEGER);
     BEGIN
          WRITELN(X):
          X:=X-1:
          IF (X)0) THEN RECURS2(X):
     END:

     PROCEDURE RECURS2:
     BEGIN
          WRITELN(X):
          X:=X-1:
          IF (X)0) THEN RECURS1(X):
     END:

BEGIN
     READLN(X):
     RECURS1(X):
     WRITELN('FINISHED'):
END.

                                PROGRAM 2c
                                ----------

         PASCAL PROGRAM TO TEST RECURSION WITHOUT PASSING PARAMETERS


PROGRAM RECURSIONWITHNOPARAMETERS;
VAR X:INTEGER;
     PROCEDURE RECURSE;
     BEGIN
          WRITELN(X);
          X:=X-1:
          IF (X)0) THEN RECURSE;
     END;

BEGIN
     READLN(X):
     RECURSE:
     WRITELN('FINISHED'):
END.
```

## BASIC RECURSION

The first high-level language to be studied was Applesoft BASIC. Here, the only way to call a BASIC routine is by using GOSUB followed by the line number at which the routine starts, and programs 1a and 1b show examples of what amount to direct and indirect recursion. Note that each routine has a condition to fulfill before the next routine is called, and this is because recursive routines with no way of stopping are not generally useful. The basic idea is to type in a number and decrease it for each level of subroutine call made in order to examine how many times the recursion has taken place. If 25 is typed in, the programs work without any problem, but any number greater than 25 will generate an 'OUT OF MEMORY ERROR' caused by over-running the stack. Examination of the BASIC stack (which is the 6502 stack area $100-$1FF) shows that for each call, the BASIC interpreter seems to use 7 bytes as opposed to the 2 bytes used by the machine code subroutine calls. Note that the stack is not allowed to recycle in this case.

It is therefore a limitation in BASIC that subroutine calls may not be nested more than 25 deep. It should be noted, however, that this restriction is not as great as it may seem because in a line-oriented language like BASIC, the GOTO statement may often be used to better advantage. For example, if in programs 1a and 1b the GOSUB statements were replaced by GOTO statements, the program would be almost identical, but work no matter how high a value is typed in.

## PASCAL

The UCSD Pascal system (developed at University of California at San Diego) requires an Apple with 64K of memory. Once a program has been compiled, the system knows exactly how much space is needed for code and data and so reserves the rest for its stack. Programs 2a and 2b are the Pascal versions of direct and indirect recursion programs, and each of them prints 'FINISHED' if a small number is inputted. However, program 2a stops with a 'STACK OVERFLOW' error after 2407 iterations, and program 2b stops after 2404 iterations. Note that as program 2b is slightly longer (and thus takes up more code), it reserves correspondingly less memory for the stack. The FORWARD declaration in program 2b is necessary as RECURS2 is called before it is defined, and UCSD Pascal has only a one-pass compiler. It should be pointed out that Pascal implementations which have compilers that make two or more passes (such as MT+86 Pascal) would not need a FORWARD declaration.

We may experiment with the stack by declaring a known amount of data to see what effect it has on the program. If we add the line VAR DUMMY:ARRAY [0..399] OF INTEGER to program 2a, we are reserving an area of memory for 4000 integers (which is 8000 bytes), and now the program stops after 1907 iterations (instead of the original 2407). It is evident that each call consumes 16 bytes of stack!

However, note that we are passing the parameter X through each procedure. Not only is this unnecessary in this case, but it also uses more stack. Program 2c is like program 2a but does not pass any arguments through via procedure calls, preferring to use X as a global variable. This program stops after 2751 iterations, showing that the simplest procedure call uses 14 bytes of stack.

```
                                PROGRAM 3a
                                ----------

              LOGO PROGRAM TO TEST DIRECT RECURSION


TO RECURSE :X
PRINT :X
MAKE "X :X-1
IF X)0  RECURSE :X
END
                                PROGRAM 3b
                                ----------

              LOGO PROGRAM TO TEST INDIRECT RECURSION


TO RECURS1 :X
PRINT :X
MAKE "X :X-1
IF X)0 [RECURS2 :X]
END


TO RECURS2 :X
PRINT :X
MAKE "X :X-1
IF X)0 [RECURS1 :X]
END
                                PROGRAM 3c
                                ----------

IDENTICAL TO PROGRAM 3a, BUT WITH AN EXTRA COMMAND AFTER THE RECURSIVE
CALL SO THAT RECURS3 IS NOT TAIL-RECURSIVE

TO RECURS3 :X
PRINT :X
MAKE "X :X-1
IF X)0 [RECURS3 :X]
FORWARD 10
END
                                PROGRAM 3d
                                ----------

        IDENTICAL TO PROGRAM 3c, BUT WITH NO ARGUMENTS PASSED THROUGH

TO RECURS4
PRINT :X
MAKE "X :X-1
IF X)0 [RECURS4]
FORWARD 10
END
```

## LOGO

The Logo language is often cited in popular magazines as being a language which enables the user to move 'turtles' around on the screen to make pretty patterns. The implementation of Logo for the Apple (which also needs 64K of RAM) is much more than this, being a list-processing language in its own right — for example, the FIRST and BUTFIRST commands in Logo are exactly analogous to the all-important CAR and CDR in LISP. In fact, Apple Logo is based on LISP and one expects to find many analogies between the two languages.

Programs 3a and 3b are the direct and indirect recursion programs, analogous to those in BASIC and Pascal, but they work no matter how high a number is typed in! This result seems to be a direct contradiction of all that was said before. It is as if the routine does not remember where it was called from.

In order to understand what is going on, let us force the routine to remember where it was called from by giving it something to do after it returns. I inserted the command FORWARD 10 as in program 3c, which moves the turtle forward 10 units. It doesn't really matter what command is put there, so long as there is one. Now, on typing RECURS3 5, the turtle moved forward 50 units after all the recursion had been done, which could only happen if each call stored where it was being called from on Logo's equivalent of stack. Of greater significance is that program 3c DOES stop with an 'OUT OF MEMORY' error if given an argument much greater than 200, while program 3a works even with 32000.

To investigate how much memory Logo uses for each subroutine call, it is necessary to know that Logo uses 5-byte quantities called 'nodes' to handle all its operations. The construction of these nodes is fairly complicated and may include pointers to other nodes, but this will not be dealt with in any depth here. However, it is possible to ask the Logo system how many nodes are available, but a meaningful answer will only be given if the system has just 'recycled' nodes it has used only temporarily (BASIC programmers call this a 'garbage collection'). The list of instructions RECYCLE PRINT NODES will display a reasonable approximation to the number of nodes the system has available.

As the number of free nodes depends on how many programs are in memory, and on exactly what has been typed in at the keyboard, a complete description of exactly how many recursions are made before running out of memory is not given. However, it can be shown that program 3c uses up 12 nodes (60 bytes!) per iteration. Program 3d does not pass any parameters through, and uses only 9 nodes per iteration.

The significance of programs 3a and 3b going on forever will be discussed in the conclusions section, but first we will perform a similar analysis of the way Apple LISP behaves — after all, LISP IS the language that Logo is based on, and it is instructive to see whether LISP also allows this infinite recursion.

## LISP

Having stated that Logo is a LISP-like language which allows the possibility of infinite recursion, we will now test OWL LISP in a similar manner. Program 4a did not go on forever, but ran out of space after 766 iterations. Note that program 4a is the analogue of the Logo program 3a which did go on forever, and it is evident that the LISP implementation is not designed to cope with this. Using the (MESSON 1) directive to print out 'garbage collection' results followed by the RECLAIM command, it can be shown that there were 26039 bytes of workspace available and therefore program 4a used 34 bytes per iteration.

As program 4a passes X as an argument through each subroutine call, program 4b (which does not pass arguments through calls) was written and this used 24 bytes per iteration.

At this stage, it should be noted that all the programs in BASIC, Pascal and Logo considered so far only made a recursion call after an 'IF' or conditional statement. Removal of the conditional statement in all these cases means that there is no natural way for the program to terminate, but does not seem to affect the number of bytes or nodes that the program uses per iteration. However, program 4c makes its recursive call without a COND statement, and this DOES seem to allow a much greater depth of recursion than program 4b — in fact each call now only uses 12 bytes. I conclude from all this that this implementation of LISP does not allow infinite recursion in the same way that Logo does, and that there are probably certain weaknesses in the implementation which case such a high memory overhead on the COND statement.

```
                         PROGRAM 4a
                         ----------

              LISP PROGRAM TO TEST DIRECT RECURSION

(DE RECURSE (X)
     (PRINT X)
     (SETQ X (SUB1 X))
     (COND
          ((GREATERP X 0)
               (RECURSE X)
          )
     )
)


                         PROGRAM 4b
                         ----------

          IDENTICAL TO 4a BUT NO ARGUMENTS PASSED THROUGH

(DE RECURS1 ()
     (PRINT X)
     (SETQ X (SUB1 X))
     (COND
          ((GREATERP X 0)
               (RECURS1)
          )
     )
)


                         PROGRAM 4c
                         ----------

              IDENTICAL TO 4b BUT WITH NO COND STATEMENT


(DE RECURS2 ()
     (PRINT X)
     (SETQ X (SUB1 X))
     (RECURS2)
)
```

## CONCLUSIONS

Having made a brief comparison of the way various languages treat recursion, it is evident that only Logo allows certain types of recursive calls to be made to an unlimited depth. This situation is not unknown in computer languages, and is called 'tail recursion'. Tail recursion is when the only recursive call in a routine is the last executable statement in that routine, and in this case a clever compiler or interpreter will know that it does not need to clutter up the stack with unnecessary return addresses. It is interesting to note that Logo allows even indirect recursion to be tail-recursive, as program 3b demonstrates, and it is evident that the implementers have gone to some trouble to ensure that this is the case. Why should this be, when even Pascal and LISP do not have this feature? What is so different about Logo that it merits such attention?

Logo is advertised as an ideal language for learning, and many instruction manuals and books (including the Apple Logo Manual itself) are full of recursive routines for drawing patterns. A particularly primitive example is the following routine to draw a square:

```
TO SQUARE
FORWARD 50
RT 90
SQUARE
END
```

Where tail recursion is not implemented, the above routine (and indeed many programs that an imaginative beginner might type in) would fairly quickly report an out-of-memory error, and frustrate or even put off the learner.

The conclusion to be drawn is that recursion and subroutine calling may only be nested to a limited depth in most of the common languages available on the Apple II, but that Logo, in spite of being a slow and even somewhat inefficient language, is not subject to such limitation if the calls are tail-recursive. This makes Logo a very good language in which to write recursive graphics.

## MAIL ORDER PROTECTION SCHEME

If you order goods from Mail Order Advertisers in this magazine and pay by post in advance of delivery, this publication will consider you for compensation if the advertiser should become insolvent or bankrupt, provided:

1. You have not received the goods or had your money returned; and
2. You write to the publisher of this publication explaining the position not earlier than 28 days from the day you sent your order and not later than 2 months from that day.

Please do not wait until the last moment to inform us. When you write, we will tell you how to make your claim and what evidence of payment is required.

We guarantee to meet claims from readers made in accordance with the above procedure as soon as possible after the advertiser has been declared bankrupt or insolvent to a limit of £1,800 per annum for any one advertiser, so affected, and up to £5,400 p.a. in respect of all insolvent advertisers. Claims may be paid for higher amounts, or when the above procedures have not been complied with, at the discretion of this publication, but we do not guarantee to do so in view of the need to set some limit to this commitment and to learn quickly of reader's difficulties.

This guarantee covers only advance payment sent in direct response to an advertisement in this magazine (not, for example, payments made in response to catalogues, etc, received as a result of answering such advertisements):

**CLASSIFIED ADVERTISEMENTS ARE EXCLUDED.**



# LUCAS MICRO

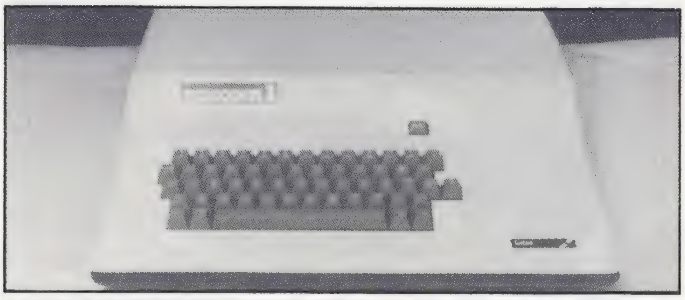
## LUCAS LX

| | |
|---|---|
| **MEMORY** | 64K RAM expandable to 256K |
| **LANGUAGE** | Microsoft BASIC |
| **CASSETTE** | 300 or 1200 baud |
| **DISC** | Single or twin 5¼ floppy disc drives<br>DOS CP/M 2.2 (supplied) or NAS-DOS |
| **KEYBOARD** | QWERTY ☑ CURSOR ☑ NUMERIC ☑ FUNCT ☑ |
| **DISPLAY** | TV ☑ MONITOR ☑ SUPPLIED ☑ |
| **INTERFACE** | PARA ☑ SERIAL ☑ BUS ☑ |
| **GRAPHICS** | BLOCK ☑ USER ☑<br>LINE ☐ RES 392 by 256<br>COLOUR 8 TEXT 80 by 25 |

**Notes.** The Lucas LX is a Z80A microcomputer aimed more at the professional and business user. Hence 5Mb Winchester disc interfacing is provided. Popular printers may be used with the RS232 serial interface, and a Centronics interface is also provided. There is an additional parallel interface connector for providing up to 16 on/off signals. The monitor supplied as standard is a 12" monochrome version: a colour monitor is also available. The high res colour graphics may be 392 by 256 in eight colours, or 784 by 256 in two colours. A wide range of applications software is available via the CP/M operating system, including Wordstar, Supercalc, and Calcstar.

## NASCOM 3

| | |
|---|---|
| **MEMORY** | 48K RAM 14K ROM |
| **LANGUAGE** | Microsoft BASIC |
| **CASSETTE** | 300 or 1200 baud |
| **DISC** | extra DOS CP/M or NAS-DOS |
| **KEYBOARD** | QWERTY ☑ CURSOR ☐ NUMERIC ☐ FUNCT ☐ |
| **DISPLAY** | TV ☑ MONITOR ☑ SUPPLIED ☐ |
| **INTERFACE** | PARA ☑ SERIAL ☑ BUS ☑ |
| **GRAPHICS** | BLOCK ☑ USER ☑<br>LINE ☐ RES 784 by 256 (two colours)<br>392 by 256 (four colours)<br>COLOUR 8 TEXT 25 by 80 |
| **SOUND** | optional |



# DEALER CBM MICRODEALER



## COMMODORE 720

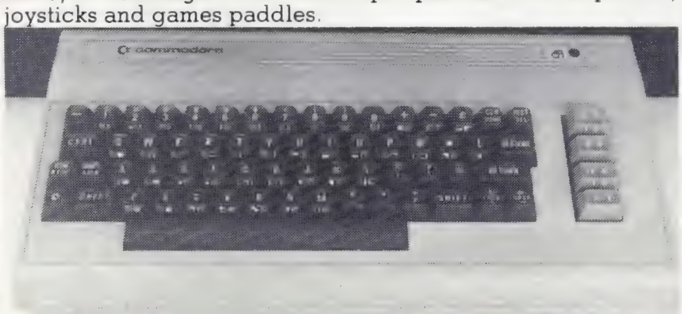
| | | | | |
|---|---|---|---|---|
| MEMORY | 256K | 20K ROM | | |
| LANGUAGE | Commodore BASIC | | | |
| CASSETTE | 300 baud | | | |
| DISC | Twin in-built floppy drives | | | |
| KEYBOARD | QWERTY ☑ | CURSOR ☑ | NUMERIC ☑ | FUNCT ☑ |
| DISPLAY | TV ☐ | MONITOR | SUPPLIED ☑ | |
| INTERFACE | PARA ☑ | SERIAL ☑ | BUS ☐ | |
| GRAPHICS | BLOCK ☑ | USER ☐ | | |
| | LINE ☐ | RES 80 by 25 | | |
| | COLOUR 16 | TEXT 80 by 25 | | |
| SOUND | **Three channels** | | | |

**Notes.** The Commodore 720 is the top model in the 700 range of business machines. It is built round the 6509 processor, but there is a dual processor (Z80 or 8088) option. The machine has been designed to meet the IEC specifications. The black-and-white monitor screen is integral and features tilt and swivel. The keyboard may be detached. The dual disc drives are built-in to the main housing and use DMA transfer, increasing speed.

## COMMODORE 64

| | | | | |
|---|---|---|---|---|
| MEMORY | 64K RAM | 26K ROM | | |
| LANGUAGE | PET BASIC | | | |
| CASSETTE | 300 baud | | | |
| DISC | extra | DOS | | |
| KEYBOARD | QWERTY ☑ | CURSOR ☑ | NUMERIC ☐ | FUNCT ☑ |
| DISPLAY | TV ☑ | MONITOR | SUPPLIED ☐ | |
| INTERFACE | PARA ☑ | SERIAL ☑ | BUS ☑ | |
| GRAPHICS | BLOCK ☑ | USER ☑ | | |
| | LINE ☐ | RES 80 by 25 | | |
| | COLOUR 16 | TEXT 40 by 25 | | |
| SOUND | **Three channels** | | | |

**Notes.** The Commodore 64 is a 6510 based micro that can also use Pascal, COMAL, LOGO, FORTH and PILOT. Programs can be loaded from cassette recorder or disc drives, both extra, or cartridges. The various peripherals include printer, joysticks and games paddles.



# SHARP MICRODEALER

## SHARP MZ-80A

| | | | | |
|---|---|---|---|---|
| **MEMORY** | 48K RAM | 4K ROM | | |
| **LANGUAGE** | Microsoft BASIC | | | |
| **CASSETTE** | 1200 baud (built-in) | | | |
| **DISC** | extra | DOS | | |
| **KEYBOARD** | QWERTY ✓ | CURSOR ✓ | NUMERIC ✓ | FUNCT ☐ |
| **DISPLAY** | TV ☐ | MONITOR ✓ | SUPPLIED ✓ | |
| **INTERFACE** | PARA ✓ | SERIAL ☐ | BUS ✓ | |
| **GRAPHICS** | BLOCK ✓ | USER ☐ | | |
| | LINE ☐ | RES 80 by 50 | | |
| | COLOUR | TEXT 25 by 40 | | |
| **SOUND** | Single channel | | | |

**Notes:** The Sharp MZ-80A is a Z80 based micro. An expansion unit, printer, floppy disc unit and other peripherals are available. Other languages can also be used such as Pascal merely by replacing the tape. With the floppy disc option the machine can respond to higher level software such as Disc BASIC and FDOS (including BASIC compiler). A small range of business and educational software is available. The supplier is **Sharp Electronics (UK) Ltd.** Thorp Road, Newton Heath, Manchester M10 9BE.

## SHARP MZ-80B

| | | | | |
|---|---|---|---|---|
| **MEMORY** | 64K RAM | 2K ROM | | |
| **LANGUAGE** | BASIC (on tape) | | | |
| **CASSETTE** | 1800 baud built-in | | | |
| **DISC** | extra | DOS | | |
| **KEYBOARD** | QWERTY ✓ | CURSOR ✓ | NUMERIC ✓ | FUNCT ☐ |
| **DISPLAY** | TV ☐ | MONITOR ✓ | SUPPLIED ✓ | |
| **INTERFACE** | PARA ☐ | SERIAL ☐ | BUS ✓ | |
| **GRAPHICS** | BLOCK ✓ | USER ☐ | | |
| | LINE ✓ | RES 320 by 200 | | |
| | COLOUR | TEXT 25 by 80 | | |
| **SOUND** | 3 channels | | | |

**Notes:** The Sharp MZ-80B is a Z80A based micro. Various other languages can be loaded as the machine is "soft", no language being fitted in ROM. Expansion unit, the MZ-80P5 printer and the MZ-80FB floppy disc drive are also available. The supplier is **Sharp Electronics (UK) Ltd.** Thorp Road, Newton Heath, Manchester.

# COMPUTING TODAY

**Lineage:** 40p per word.
**Semi display:** £9.00 per single column centimetre
Ring for information on series bookings/discounts.

All advertisements in this section must be prepaid.
Advertisements are accepted subject to the terms and conditions printed on the advertisement rate card (available on request).

# 01-437 0699

**Send your requirements to:**
**MARK BECAREVIC**
**ASP LTD, 1 GOLDEN SQUARE,**
**LONDON W1**

## ACCESSORIES

### BLANK CASSETTES!

**TOP QUALITY PROFESSIONAL BRAND COMPUTER/AUDIO CASSETTES AT BUDGET PRICES**

Packed in boxes of 10 cassettes complete with labels, inlay cards and library cases. Prices include VAT post & packing

| Length | Box Price (10) | Qty | Amount |
|---|---|---|---|
| 5 min | £4.85 | | |
| 10 min | £4.90 | | |
| 12 min | £4.95 | | |
| 15 min | £5.00 | | |
| 30 min | £5.20 | | |
| 60 min | £5.80 | | |
| 90 min | £7.50 | | |

Cheque/Postal Order enclosed for £
NAME ..........................................
ADDRESS ......................................
..................................................

**PROFESSIONAL MAGNETICS LTD**
**Cassette House, 329 Hunslet Rd, Leeds.**
**Tel: (0532) 706066**
TRADE ENQUIRIES WELCOME

### MRL PRINTERS

Chinwa CP80 195 (224 inc VAT). Star Gemini 10X (120 cps 80 col) 215 (247 inc VAT). Star Gemini 15X (120 cps 132 col) 335 (385 inc VAT). Star Delta 10 (160 cps 80 col) 359 (413 inc VAT). Juki Daisywheel (18 cps) 369 (424 inc VAT). Plus paper, ribbons & refills etc. Free delivery by Securicor.

**RING 0506 31605 FOR DETAILS**

**MRL Printer Interfaces**

Available to fit most micros: Electron, Spectrum, Atari, Commodore 64 incl free cable, free delivery. From **£39.95** inc VAT

**RING 0506 31605 FOR DETAILS**

Selection of Disc Drives at bargain prices from **£135** (inc VAT) to fit BBC, Tandy, Nascom etc

**MICRO RESEARCH LTD (Freepost)**
**Industrial Unit 6, Knightsridge East, Livingstone EH54 5BR, West Lothian.**

### CHILTERN ELECTRONICS

**Amazing Value in Video Monitors**

Professional quality equipment by AGC Corp ideal for your micro. Look at these features ★ Superb resolution for graphics 80 col. etc ★ Bandwidth 10hz-22Mhz ★ Green Screen P31 phorsphor ★ Antiglare tube ★ Attractively styled case ★ Composite video mains operation. Available in 9" Model HM911 **£85.** 12" Model HM123 **£89.**

**FANTASTIC KEYBOARD BARGAIN!!** Scoop purchase allows us to offer a 49 key QWERTY layout, ASCII encoded keyboard. Fully cased, used but guaranteed working. Full logic diagrams and information supplied. **ONLY £15.** Add carriage £1.85 and VAT Educational discounts. Access welcome Same day despatch. Call Gary Kent on 02407 71234. **High Street, Chalfont St Giles, Bucks HP8 4HQ. Telex: 262284**

# ASP CLASSIFIED 01 - 437 0699

**Quality Cassettes/Disks/Paper**

| | |
|---|---|
| 5 x C15 Cassettes | **£3.00** |
| 10 x C15 Cassettes | **£5.60** |
| 5 x C20 Cassettes | **£3.20** |
| 10 x C20 Cassettes | **£6.00** |

5 x 5¼ Floppy Disks **£7.50**. Listing Paper (11" x 9.5") 500 Sheets only **£5.00**
Labels, Cases & Postage Inclusive.
Cheques, PO's to:
C & S COMPUTER SERVICES,
144 Sutcliffe Avenue, Grimsby, South Humberside DN33 1AP.

## SOFTWARE APPLICATIONS

### POOLS PREDICTION

**"POOLSWINNER"**

The most sophisticated Pools Prediction Aid available. Gives probabilities of score draws, draws, homes or aways, based on two databases holding over 20,000 matches (included). The databases are automatically updated as results are entered. This year's results are supplied with the package. Can be used in simple mode, or with parameter adjustments to develop your own unique forecast method. Fully documented, available now for Apple, Spectrum (48K), Dragon, ZX81 (16K), Commodore 64, BBC (B) (others — please enquire).

**£15.00** (discs/tapes)

**"POOLSDATA"**

Complete record of all English Football League matches 1978-83. Team, scores and dates of 10,000 matches held in simple format, ready for your analysis. Starter analysis programs and full documentation included. Available for Apple, Spectrum, ZX81, BBC, Dragon, Commodore series.

**5 yrs Data £15.00 — 2 yrs Data £7.50**

**SELEC SOFTWARE (CT), 37 Councillor Lane, Cheadle, Cheshire. 061-428 7425**

### CORTEX — FORTH

Full fig-forth with extensions for power tran cortex computer. Supports Disc and cassette, 40 page manual. 16K supplied in two 2564 Eproms. Replaces 1st two Basic Eproms. **£35 inclusive**.

**LOMBARD SYSTEMS**
**18 Lombard Street, Lidlington, Bedford MK43 0RP.**

**SPECTRUM BIGPRINT.** 24 heights, 32 widths, £2.50. K. Stock, 132 Paynes Lane, Coventry.

## SOFTWARE APPLICATIONS

**COMMODORE 64.** "Spriter". Enables single or multicolour sprites to be designed interactively, complied and saved to tape. Tape £4.60. 43 Chanctonbury Drive, Shoreham, Sussex BN4 5FR.

**ORIC FORTH** Floating Point extension words. Full facilities, Trig functions, complex numbers and Turtle graphics. £4.50 (including full documentation). David McKelvie, 86 Northcott, Bracknell, Berks.

**TRS-80 V. GENIE LII.** Add fifteen new program commands to your Basic, with our extender utility program. Machine Code. £10.50. Send to: Olympic Software, 15 Arnhem Close, Aldershot, Hants GH11 1RJ.

**SPECTRUM TAPE COPY.** Full assembled and commented machine code listing with Hex loader and full instructions. Cheques etc to S. Woodthorpe, 64 Abingdon Avenue, Lincoln.

## BOOKS AND PUBLICATIONS

**PARAPHYSICS** Journal (Russian Translations); Psychotronics, Kirlianography, Heliphonic Music, Telekinetics, Computer Software. SAE 4 x 9". Paralab, Downton, Wiltshire.

### NOW ON SALE

**PEEKING AND POKEING THE SHARP MZ700 SERIES**

Includes additions to S-Basic and copying, data handling, peeks and pokes, Interpreter points, V Ram plus second character set, plus more.

Available from most Sharp stockists **£6.95**, or direct. Inc P&P from D. C. Brennan Eng, 14 North Western Ave., Watford Herts. (Dept. CT). Dealer enquiries welcome on 04243 2376

Peeking and Poking the MZ 80 K
Peeking and Poking the MZ 80 A
Still available at £4.30

## SERVICES

### ELIMINATE FAULTY CASSETTES

DataClone is the first company in the UK established specifically for the duplication of data cassettes.

All other duplicating houses are audio orientated — only DataClone has a duplicating system designed from scratch purely to handle computer information.

The result?

Greatly improved reliability in data transfer rates from 300 to beyond 2400 baud — previously unattainable.

All formats catered for. Quantities from 100 to infinity.

Contact us now for brochure.

DataClone — the first specialist service for computer cassettes.

### DATACLONE

**Unit 1, Roslin Square, Roslin Rd., Acton, London W3.**
**Tel: 01-993 2134 Telex: 21879**

## FOR SALE

**VIDEO GENIE EG3003** (incl. cursor keys and cassette) 16K. £100 + P&P. Tel: 0642 460766 after 6pm.

**TANDY TRS80** Level II. TRS80 interface, line printer IV, CTR80 cassette, v.g.c. £500 ono. Telephone Leicester 857360.

# COMPUTAMART

AT A GLANCE...AT A GLANCE...AT A GLANCE...AT A GLANCE...AT A GLANCE...AT A GLANCE...

## BERKSHIRE

Micro General

PRINTER SPECIALISTS

Call for advice on printer selection.
Epson & Microline stockists. Interfaces & cables available for all popular micros.
6 THE BIRCHWOODS, TILEHURST, READING.
TEL: 0734 25226

## CHESHIRE

northern computers

Churchfield Rd., Frodsham.
Tel: (0928) 35110

Open: 6 days 9-6. Retail and Wholesale. Apple II & III, Atom, BBC, VIC20/64, Newbrain, Dragon 32, Electron, Spectrum, All accessories. Easy parking off M56.

Computer Junk Shop

We Buy, Sell, Break Computers & Peripherals.
10 Waterloo Rd, Widnes, Halton. Tel: 051 420 4590.

## CORNWALL/DEVON

A. B. & C. COMPUTERS (CT)
Duchy House, 6 Lower Aylmer Sq., St. Austell.
Tel: 0726 64463/67337

Wide range of popular Micros, Printers, books and accessories. We stock all U need — try us first for service and competitive prices

LOOKING FOR MICRO-COMPUTER HARDWARE OR SOFTWARE?
THEN LOOK NO FURTHER THAN COMPUTAMART

## DORSET

ZIPPY ELECTRONICS
West Dorsets Nascom Dealer

Business and Personal Computer Systems, Printers, Software. Tel: Bridport (0308) 56539

Mail Order Service operated.
Access/Barclaycard accepted.

## HERTFORDSHIRE

NEWBRAIN & SANYO
HARDWARE & SOFTWARE

Printers, Epson, KDC, Shinwa, Juki etc. Monitors, Tape Recorders, Books, Expansions. Ask for details of Sanyo 555. Mail Order.
ANGELA ENTERPRISES
Tel: Stevenage (0438) 812439 anytime

## NORTH KENT

MEDWAY COMPUTERS
141 NEW ROAD, CHATHAM.
Tel: 0634 826080

ACORN COMPUTER

Open Mon-Sat 10 - 5. Closed Weds.
Most computers and software stocked.

## LANCASHIRE

LANCASHIRE MICROS
51 QUEEN STREET, MORECAMBE.
Tel: 411435. Also open Sundays.
Sinclair, Commodore, Acorn/BBC + Dragon.
Authorised dealer for Bug-Byte, Imagine, Quicksilva. Artic, Melbourne House, Silversoft etc.

BUG-BYTE AUTHORISED DEALER

LEIGH COLOUR LABORATORY LTD
87 Chapel St., LEIGH,
Tel: 0942 607661
Open: 9 — 5.30.

NSC COMPUTER SHOPS
29 Hanging Ditch, Manchester.
Tel: 061 832 2269
Open: Mon-Fri 9.30am-5.30pm.
Sat 10-5. Retail and Wholesale.

## LINCOLNSHIRE

SHARP CENTRE
16 Melville Street,
Lincoln.
Tel: Lincoln 32379.
Open: 9am-5.30pm closed Wed.

SHARP

## LONDON

HENRY'S COMPUTER SHOP
404-406 Edgware Road, London W2 1ED.
Tel: 01-402 6822
Open: 6 days a week. Order by phone or call in and see for yourself.

A. J. Duchesne
(Computer Consultants) Limited
Specialists in Small Business Computing
10-12 Creechurch Lane, London EC3A 5AY
Telephone: 01-621 0433

COMPUTING +
264 EARLS COURT RD., LONDON SW5
Tel: 01 - 373 4508
Supplers of all Pete & Pam products and a large software selection for all micro computers.

## SOUTH LONDON

CROYDON COMPUTER CENTRE

CCC

Authorised Acorn Service Centre
29a Brigstock Rd., Thornton Heath, Surrey. Tel: 01 - 689 1280
BBC, Acorn, Electron, Genie, Oric, Kaga Microvitek Zenith Monitors. OKI 80, 82A + 84 Printers. Paper, Ribbons, Software etc. BUY-HIRE.

## MIDDLESEX

L.B. ELECTRONICS
11 Hercies Rd, Hillingdon.
Tel: Uxbridge 55399 (24hr ans. service)
Open: 6 days, 9.30am-6pm, (lunch 1-2.15 except Sat) Surplus equipment, memory, EPROMs etc. Also established mail order service.

## NORFOLK

ANGLIA COMPUTER CENTRE
88 St Benedicts Street,
Norwich.
Tel: (0603) 29652/26002.
Open: 6 days 9am-5.30pm.

## NORTHAMPTONSHIRE

NORTHAMPTON
HOME COMPUTER CENTRE
58A Wellingborough Road, Northampton.
Tel: (0604) 22539
Open: 6 days a week from 10 - 6

## NORTHERN IRELAND

NEWBURN Ballycorry Co. Antrim
6 Days until 8pm. Gemini, Galaxy, Nascom, Acorn Computers. Discs, monitors, printers, servicing, hire. Industrial Control. Accounts, Word Processors etc.
WHITEHEAD 78330

## SCOTLAND

VICTOR MORRIS GLASGOW
TANDY TRS 80, VIC 20, VIDEO GENIE, APPLE PANASONIC, CUMANA, EPSOM ETC.
340 Argyle Street, Glasgow G2: 041 221 8958

LOOKING FOR MICROCOMPUTER HARDWARE OF SOFTWARE?
THEN LOOK NO FURTHER THAN COMPUTAMART

## SUSSEX

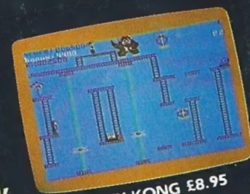
GAMER
24 Gloucester Road, Brighton.
Tel: 0273-698424.
Open: Mon-Fri 10am-5.30pm,
Sat 9am-5.30pm.

## TYNE AND WEAR

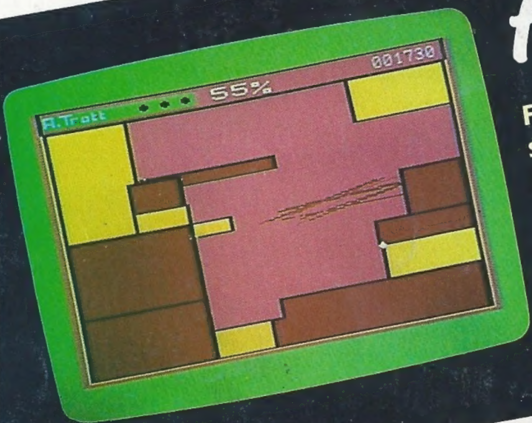
HCCS ASSOCIATES
533 Durham Rd., Low Fell,
Gateshead. Tel. Newcastle 821924.
Open: 6 days 9am-5.30pm (Sat 10am-5.30pm). Specialists in: Acorn, BBC, Video Genie, VIC 20.

# SUPERSOFT

## the name to remember

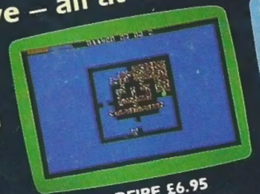


CRAZY KONG £8.95

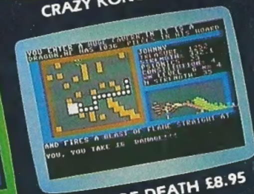


## for games

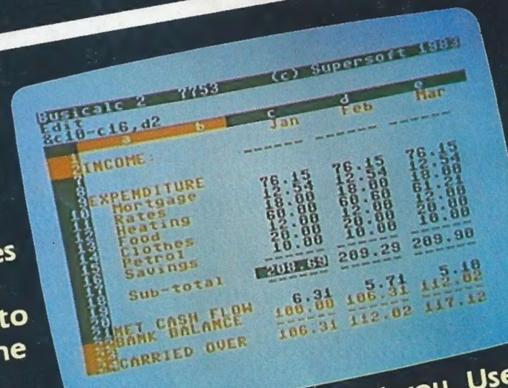
For only £8.95 you can buy a game that's exciting, soothing, and frustratingly addictive – all at the same time! STIX looks so different and sounds so different from all those other games that it will seem like being in another dimension when you sit down to play.



WILDFIRE £6.95



HALLS OF DEATH £8.95

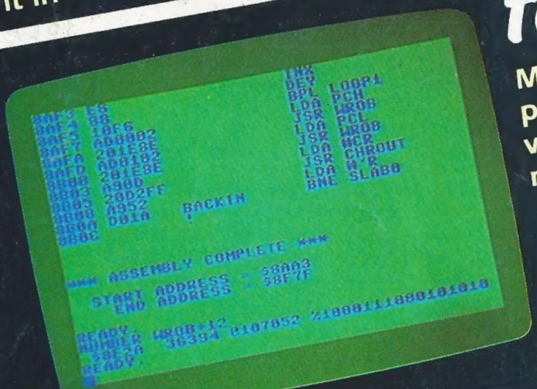


## for business

Show your computer who's master with BUSICALC! Spreadsheet programs are used by large and small businesses to juggle with figures, prepare reports and so on. Some are very powerful indeed. The problem is that they're difficult to learn, and tricky to use – which is why we came up with the BUSICALC series.

Whether you choose BUSICALC 1, BUSICALC 2, or BUSICALC 3 you'll get a program you can understand – and one that almost seems to understand you. Use it in the home, use it for teaching, use it at work – it'll save you time and money.

## for programmers



MIKRO is a full 6502/6510 ASSEMBLER with the power that professional programmers need, yet so simple to use that we recommend it to beginners! The MIKRO cartridge has many other facilities including editing commands and a machine language monitor, all for £57.50.

**There's much more for the 64 in the SUPERSOFT catalogue. Ask your computer dealer for a copy, or phone 01-861 1166.**

## The Best 64 Software

SUPERSOFT, Winchester House, Canning Road, Wealdstone, Harrow, Middlesex, HA3 7SJ Telephone: 01-861 1166